Inserción de Bitflips en Mapas de Configuración de FPGAs

Autor: Victor Alaminos Beneitez
Directora: Hortensia Mecha
Sistemas Informáticos
Universidad Complutense de Madrid
Facultad de Informática
Curso: 2009/10

Facultad de Informática

Quiero agradecer con este trabajo a mi directora por la paciencia que ha tenido conmigo y la gran ayuda y apoyo que me ha aportado en esta dura recta final de la carrera.

También quiero agradecer a mis padres y a mi novia por el apoyo ofrecido y la paciencia que han tenido para soportarme este duro año.

Palabras Clave del Proyecto

FPGA, hardware reconfigurable, SEU, bitflip, inyección de errores, mapa de bits, bitstream, memoria de configuración, reconfiguración parcial, Virtex II Pro, Virtex II Configuration Viewer, Virtex II Configuration Comparer, Virtex II Partial Reconfiguration, Nessy 2.0.

Autorización del Proyecto

Autorizo a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Victor Alaminos Beneitez

Resumen

Hoy en día se pretende enviar satélites al espacio capaces de realizar múltiples tareas con objetivos muy diferentes. Sin embargo, con el fin de minimizar gastos, estos dispositivos deben ocupar el mínimo espacio posible, con lo que la variedad de aplicaciones y el tamaño de los dispositivos parecen objetivos contrapuestos. Las posibilidades que ofrece el hardware reconfigurable parece una solución a dichos problemas.

En este trabajo se ha contribuido al desarrollo de una plataforma de inyección de errores que simula los SEUs, con el fin de medir la robustez de los circuitos. La idea es poder usarla para comparar distintas soluciones de protección de los circuitos implementados sobre hardware reconfigurable.

Las posibilidades de reconfiguración parcial y dinámica de las FPGAs hacen de esta tecnología una solución barata y fácil. Sin embargo, al depender su funcionalidad de la memoria de configuración, los circuitos implementados son vulnerables a los llamados Single Event Upsets (SEU), inducidos por radiación, que pueden alterar su comportamiento. En particular, se han implementado tres herramientas, Virtex II Configuration Viewer, para poder leer un bitstream de configuración y presentar sus comandos de configuración de una forma clara; Virtex II Configuration Compararer, para comparar entre dos bitstream y detectar cuales son sus diferencias; Virtex II Partial Reconfiguration, que sirve para crear bitstreams de configuración parcial en los que dado un bitstream original se modifica un solo bit, con el fin de simular el efecto de un SEU. Esta última se ha integrado en la plataforma de inyección de errores que va alterando 1 a 1 todos los bits de configuración y comprobando si cambia la ejecución de un determinado conjunto de entradas.

Abstract

Nowadays, it is pretended that the satellites put into orbit are able to perform multiple tasks with very different objectives. However, in order to minimize the costs, this devices have to take up the lesser space as possible, thus the variety of applications and its size seems to be opposed. The possibilities that reconfigurable hardware offers seems to be a solution to those problems.

In this task, we have contributed to the development of an error injection platform which simulates SEUs in order to measure how robust the circuits are. The idea is to be able to use it for compare distinct solutions of protection of circuits developed in reconfigurable hardware.

The possibilities of partial and dynamic reconfiguration of the FPGAs make this technology a cheap and easy solution. However, because of its dependence on its configuration memory functionality, the implemented circuits are vulnerable to the Single Event Upsets (SEU), induced by radiation that may alter its performance. In Particular, three tools have been implemented: Virtex II Configuration Viewer, in order to been able to read a bitstream configuration and present its configuration commands in a easy way; Virtex II Configuration Comparer, in order to compare two bitstream and detect which are their differences; Virtex II Partial Reconfiguration, which creates bitstreams of partial configuration which modifies just one bit from a given original bitstream in order to simulate the effect of a SEU. This last one has been integrated within a platform of error injections that alters 1 by 1 every single bit of configuration and checking if the execution of a specific group of entries changes.

Índice

1. Introducción

Debido a la gran evolución de la tecnología aeroespacial y el incremento del tiempo de vida de los satélites más allá de los 10 años, los estándares que se han utilizado hasta ahora han quedado anticuados y la reprogramabilidad en tiempo de ejecución se hace totalmente necesaria. El uso de Aplication Specific Integrated Circuit (ASIC) o circuito integrado de aplicación específica, que ha sido diseñado para un propósito o aplicación específica es un obstáculo para la mejora de las aplicaciones implementadas en un satélite, puesto que una vez diseñado el circuito no se permite la reprogramabilidad del diseño. Si se necesita añadir una nueva funcionalidad al circuito, es necesario la construcción del ASIC completamente, con lo que conlleva un nuevo diseño, en el que los diseñadores invierten muchas horas de trabajo y una nueva construcción, con el respectivo coste de fabricación. Una posible solución es utilizar Reprogrammable (SRAM) Field Programmable Gate Array (RFPGA). Las FPGAs son muy útiles debido a su bajo coste, su capacidad de reprogramación y el poco espacio que ocupan sus circuitos.

Sin embargo, las FPGAs presentan un problema, por su sensibilidad a los llamados Single Event Upsets (SEU) inducidos por radiación. Por ejemplo, en el espacio una partícula solar puede chocar con la superficie de la FPGA debido a la ausencia de la protección atmosférica que posee la Tierra. Esto puede provocar una alteración en la memoria de configuración y biestables de estado, lo que puede implicar una modificación de la funcionalidad del circuito.

Distintos grupos de investigación están desarrollando métodos para resolver este problema. Una solución es construir las FPGAs con materiales especiales o revestirlas de alguna forma para protegerlas y así mitigar la radiación. Esta solución es la más cara, pero no permite resolver el problema en caso de que un SEU llegue a alterar el circuito. Otra solución es encontrar la protección por diseño de los propios circuitos, incluso posibilitar la recuperación en caso de detección de un fallo y así disminuir su vulnerabilidad.

Este proyecto se encuentra dentro de un proyecto más ambicioso que pretende generar una plataforma de inyección de errores para medir la calidad de cada diseño frente a posibles SEUs. En particular, nos centraremos en la inserción de bitflips en el mapa de configuración de la FPGA para emular la alteración de la memoria de configuración por una partícula solar. Es decir, alteraremos un bit del mapa de bits de configuración de la FPGA para luego cargar esta nueva configuración en la FPGA y observar si esto ha producido cambios en su funcionalidad.

2. Configuración de las FPGAs

Las Field Programmable Gate Array (FPGA) son dispositivos que poseen bloques de lógica programable, así como su interconexionado y funcionalidad también es programable. Son muy útiles debido a que se pueden programar complejos circuitos secuenciales o una simple puerta lógica. Existen otros tipos de dispositivos con los que también se pueden implementar circuitos digitales. Por ejemplo mediante microcontroladores con un conjunto fijo de instrucciones, DSPs, ASIC, o CPLDs basados en memoria flash.

Sin embargo hay aplicaciones en las que usar este tipo de dispositivos no es suficiente, y el uso de FPGAs es totalmente necesario. Esto es gracias a la ventaja que poseen las FPGAs, que son dispositivos reconfigurables, tienen un menor coste que otro tipo de dispositivos y a las vez se ejecutan más rápido que otros dispositivos reprogramables. Otra ventaja es que permiten extraer el máximo paralelismo, ya que lo que se implementa son circuitos digitales directamente en hardware. Esto no sería posible en un microcontrolador, porque ejecuta instrucciones de forma secuencial.

2.1. Método de Configuración y Reconfiguración

La configuración de una FPGA tipo Virtex II se realiza mediante frames verticales de un bit de ancho y de largo igual a la altura del dispositivo. Estas frames son la mínima unidad de configuración direccionable, es decir, la configuración de una FPGA solo se puede realizar indicando la dirección de la frame que se quiere configurar, para después configurarla entera de arriba a abajo.

El número de frames necesarias para configurar una FPGA varía según el tipo de dispositivo. En el caso de la Virtex-II Pro que se utilizó en este proyecto (XC2VP30), necesita 1.756 frames de configuración y cada frame tiene una longitud de 206 palabras de 32 bit cada una. Esto hace un total de 11.575.552 bits de configuración. El cálculo es trivial:

$$1756\ frames * (206\ palabras * 32\ bits) = 11575552\ bits$$

Para realizar una configuración total de la FPGA no hace falta ir indicando una a una la dirección de cada frame. Simplemente se configuran todas las frames de la FPGA una tras otra directamente desde la primera hasta la última introduciendo la configuración que se desea. Finalmente se envía el comando de reiniciar la ejecución de la placa.

Como se mencionó anteriormente, este tipo de FPGA son reconfigurables dinámicamente, es decir, una vez que ya se ha configurado la FPGA se puede reconfigurar parcialmente mientras las aplicaciones implementadas siguen ejecutando. La reconfiguración total seria similar a una configuración normal de una FPGA. Una reconfiguración parcial posee la peculiaridad de que se puede configurar varias o incluso una sola frame del mapa de bits minimizando el tiempo de configuración. Esto es posible gracias a que el interfaz de configuración se mantiene activo para que poder recibir datos de reconfiguración. Para ello, en la configuración inicial de la FPGA se tiene que haber indicado la opción de persistencia, para que la placa permita reconfigurarse en ejecución. Para configurar la frame hay que indicar la dirección de la frame que se quiere configurar y luego se configura la frame entera de arriba a bajo. Sin embargo, se asegura que si un bit de una frame no se modifica, en la reconfiguración de dicha frame, ese bit no sufre glitches, y el circuito implementado correspondiente no se ve alterado en ningún momento. Por tanto, si si solo se cambia un bit, solo ese bit puede influir en el cambio de comportamiento del circuito durante la ejecución.

Además, en reconfiguración parcial no se envía comando de reinicialización de la FPGA, para que siga ejecutándose como si nada hubiese cambiado. Cosa que no ocurría cuando se configuraba la FPGA completamente.

2.2. Paquetes del Bitstream

Para configurar una FPGA se carga en esta un bitstream o mapa de bits. Este bitstream está compuesto de una palabra de sincronización de 32 bits (0xAA995566) y numerosos paquetes de datos. La palabra de sincronización sirve para alinear los dispositivos lógicos de configuración de la FPGA con el primer paquete de datos del bitstream. Los paquetes de datos del bitstream consisten en una cabecera de 32 bits y un cuerpo de longitud variable. Existen dos tipos de paquetes:

1. El tipo 1, utilizado para paquetes pequeños, hasta $2^{11}-1$ palabras (Figura 1).
2. El tipo 2, utilizado para paquetes largos, hasta $2^{27}-1$ palabras (Figura 2).

Type			WR	RD	Register Address													RSVD		Word Count										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x

Figura 1: Cabecera de paquete Tipo 1

Type			WR	RD	Word Count																									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura 2: Cabecera de paquete Tipo 2

En la cabecera se indica qué comando se quiere realizar, ya sea escribir o leer y en qué registro del dispositivo. También se pueden encontrar comandos que no configuran nada y que únicamente sirven para liberar los buffers del último comando que se ha procesado, son las NOOPs (0x20000000).

Existen una gran cantidad de registros en la FPGA, solo ciertos registros del dispositivo se pueden leer y escribir, otros solo se pueden leer o escribir. Cada registro está identificado por una dirección para poder configurarlos. La funcionalidad de dichos registros viene explicada con más detalle en el Apéndice 1 en el apartado de configuración de la FPGA del tipo Virtex II. Entre estos registros nos centraremos en los registros CRC, FAR, FDRI, CTL y COR.

1. El registro Cycle Redundancy Checks (CRC) almacena una palabra de 16 bits de CRC. Este registro se puede tanto escribir como leer. Se puede acceder a él directamente indicándolo en un comando o en el paquete FDRI la ultima palabra sirve para escribir un CRC automático. Así con esta palabra de CRC se comprueba la consistencia de los paquetes FDRI que se envían a la FPGA. Más adelante, en el proyecto, esta función de comprobación del CRC habrá que desactivarla para que se pueda modificar e insertar bitflips en el mapa de bits de configuración sin que se produzcan errores de CRC.
2. El registro Frame Address Register (FAR) se configura para indicar cuál es la dirección de la frame que se quiere configurar. Importante para la reconfiguración parcial, para determinar donde se pretende insertar el bitflip.
3. El registro Frame Data Input Register (FDRI) sirve para cargar las frames de configuración. Cada vez que se llena su contenido, se vuelca la frame en memoria en la dirección indicada por el registro FAR, y ésta se incrementa automáticamente en 1. En este registro es en el que más se trabajará, ya que es el que se encarga de cargar el mapa de bits de configuración donde modificaremos 1 bit para producir un efecto similar a un SEU.

4. Para configurar el nivel de seguridad se utiliza el Control Register (CTL). Este registro será necesario para indicar la persistencia de la FPGA, que al estar activa, permitirá reconfigurar la FPGA mientras este funcionando, que nos será necesario para poder reconfigurar parcialmente.
5. Para determinar ciertas configuraciones del dispositivo, como la revisión del CRC, se hace mediante el Configuration Options Register (COR). Existen muchos tipos de configuraciones, nos centraremos en deshabilitar la comprobación del CRC, dado la necesidad de obviarlo por las razones que se explicaron anteriormente.

Un ejemplo de los principales comandos de configuración del mapa del bits de configuración de una FPGA del tipo Virtex II sería:

1. Sincronización de la anchura de palabra del bitstream con el dispositivos
2. Reset del CRC
3. Indicar la longitud de la frame
4. Configurar las distintas opciones de configuración
5. Comprobación del ID del dispositivo que se va a configurar
6. Escribir en el registro FAR la dirección de la primera frame
7. Escribir en el registro FDRI todas las frames correspondientes al circuito (completo o modificado) junto con el AutoCRC
8. Frame de instrucciones NOOPs para vaciar los buffers del registro FDRI
9. Configuración de las opciones de seguridad
10. Desincronizacición con el dispositivo
11. Finalización de la configuración con instrucciones NOOPs

1	AA995566	Sync Word
2	30008001 00000007	Write 1 Words to CMD RCRC Command
3	30016001 000000CD	Write 1 Words to FLR Frame Length Register = 0x000000CD Words

4	30012001	Write 1 Words to COR
	00053FE5	Configuration Options Register = 0x00053FE5
5	3001C001	Write 1 Words to IDCODE
	0127E093	Device ID = 0x0127E093 (XC2VP30)
6	30002001	Write 1 Words to FAR
	00000000	Frame Address Register = 0x00000000
7	30004000	Type 1 Write 0 Words to FDRI
	500585D6	Type 2 Write 361942 Words to FDRI
	00000000	Word 1 Frame Data Input Register = 0x00000000
	…	…
	00000000	Word 361942 Frame Data Input Register = 0x00000000
	00007191	AutoCRC Word = 0x00007191
8	20000000	NOOP Word 1
	…	…
	20000000	NOOP Word 206
9	3000A001	Write 1 Words to CTL
	00000000	Control Register = 0x00000000
10	30008001	Write 1 Words to CMD
	0000000D	CMD Register = DESYNCH Command
11	20000000	NOOP Word 1
	20000000	NOOP Word 2
	20000000	NOOP Word 3
	20000000	NOOP Word 4

Otros comandos importantes del bitstream de configuración son los comandos de GRESTORE, para resetear los IOBs y flipflops de los CLBs a sus valores iniciales y el comando START, para iniciar la secuencia de startup de la FPGA. Estos dos comandos solo aparecen en la configuración total de la FPGA ya que conllevan el reset de sus componentes y la iniciación de la ejecución de la FPGA. Comandos que no se necesitan en la reconfiguración parcial ya que no desempeñan funcionalidades deseadas en la reconfiguración parcial.

30008001	Write 1 Words to CMD
0000000A	CMD Register = GRESTORE Command
30008001	Write 1 Words to CMD
00000005	CMD Register = START Command

3. Incorporación de Bitflips en una Plataforma de Inyección de Errores

3.1. Estudio del Mapa de Bits para un Diseño Determinado

Para realizar el estudio del mapa bits de configuración de la FPGA, se realizó el diseño de un contador de 4 bits, y se implementó la herramienta **Virtex II Configuration Viewer** con la que se podía observar y entender de forma más clara el mapa de bits de configuración. Una vez que somos capaces de entender la estructura global de un mapa de bits, nuestro siguiente problema es encontrar la forma de alterar un punto concreto del circuito, modificando el mapa de bits correspondiente. Es decir, dado un SEU en una posición física determinada de un circuito, encontrar el bit del mapa de bits que hay que modificar para producir el mismo efecto.

Para encontrar los interfaces que configuran los comando del bitstream en el dispositivo se hizo uso de la herramienta FPGA Editor, que posee el ISE de Xilinx. Con FPGA Editor, además de visualizar la posición física de cada uno de los elementos de un circuito, se pueden modificar los valores iniciales de los registros del dispositivo o la funcionalidad de las LUT y crear un archivo de configuración (.bit) con esos valores. Por lo que si se modificaba con FPGA Editor las características de una LUT en una determinada posición y luego se creaba el bitstream correspondiente, tendríamos el archivo bitstream original sin ninguna modificación, y otro archivo bitstream modificado mediante el FPGA Editor que es idéntico al anterior salvo la frame correspondiente al bit modificado, y el comando de AutoCRC. El comando de AutoCRC no daba problema por que se autogeneraba con la modificación de FPGA Editor.

Encontrar en el mapa de bits la palabra donde se encontraba la modificación que se había insertado en el bitstream, era un trabajo prácticamente imposible manualmente. Para la ayuda de este proceso de localización se realizó una aplicación que pudiese leer dos bitstream que fueran compatibles con la Virtex-II Pro mediante dos archivo de bytes (.bit), y crease un informe de las diferencias de configuración entre los dos bitstream, esta herramienta es **Virtex II Configuration Comparer**.

3.2. Incorporación de Bitflips en Nessy

Llegando a este punto se pusieron en común nuestros estudios y el trabajo realizado junto con otro proyecto que estaba trabajando en paralelo, “Inserción y detección de errores en dispositivo FPGA” de los autores Carlos Sánchez-Vellisco, Antonio José García y David Fernández. En este proyecto se estaba realizando la herramienta “Nessy 2.0” de detección de errores. Esta herramienta servía para cargar un bitstream en la FPGA, y ejecutar un banco de pruebas. Con este proceso se creaba un fichero “Golden” que serviría para comparar en posteriores ejecuciones los resultados de todas ellas.

Una vez que se hacia la primera ejecución y se generaba el fichero Golden, se ejecutaba de nuevo, y antes de que terminase la ejecución se podía parar, pudiendo cargar otro bitstream modificado en un bit con el FPGA Editor o manualmente. Si la modificación se había realizado en un bit que afectaba al circuito implementado, cuando se reanudaba la ejecución, Nessy 2.0 detectaba que no coincidían las ejecuciones y que un fallo en la FPGA se había producido y debía ser reparado. Si se cargaba otra vez el bitstream original y se ejecutaba de nuevo, Nessy 2.0 indicaba que la ejecución había sido correcta y sin errores.

Para la creación de errores manualmente había que tener en cuenta el AutoCRC, por lo que si se quería realizar una modificación de un bitflip manualmente había que desactivar la comprobación de CRC en el comando COR. En el bit 29 del registro COR había que poner su valor a “1” para deshabilitar la comprobación de CRC.

Entonces se procedió a realizar distintas pruebas implementando un contador en la FPGA, se cargaba el bitstream del contador con Nessy 2.0 y se generaba el fichero Golden, luego se modificaba el bitstream original en algún sitio al azar con el FPGA Editor. Se cargaba el nuevo bitstream modificado y se comparaba la ejecución a ver si ocurría algún error. Si no ocurría ningún error era por que se había modificado algo fuera de lo que era el circuito que se estaba probando. Por el contrario si se modificaba por ejemplo el valor de inicio de los registros utilizados por el contador y luego se cargaba el bitstream completo con reinicio de la placa, Nessy 2.0 detectaba un mal funcionamiento. Al igual si se modificaba la funcionalidad de las LUT Nessy 2.0 detectaba un mal funcionamiento, necesitando volver a cargar el bitstream original y restaurar la correcta ejecución del contador.

Sin embargo, dado que la carga de un bitstream completo, además de que detiene la ejecución de toda la FPGA, tarda bastante tiempo, se procedió a realizar un estudio parecido al anterior con el mismo contador. En este caso en vez de cargar el mapa de bits completo, se reconfiguraba parcialmente solo la frame en la que insertábamos el bitflip. Para que la reconfiguración parcial funcionase había que indicar en el bitstream de la configuración inicial que se activase la persistencia. Esto se podía realizar de dos maneras, o indicarlo en las opciones al generar el bitstream mediante Nessy 2.0, o modificar el bit 3 del registro CTL a "1" para activar la persistencia. Primeramente se realizaron pruebas manualmente mediante FPGA Editor modificando la funcionalidad de las LUT y comparando la ejecución con Nessy 2.0 como se explicó anteriormente. Sin embargo este no era un proceso muy automático de detección de errores por lo que se llevó a cabo la programación de otra herramienta, la cual se integra en Nessy 2.0 para automatizar el proceso anteriormente descrito, **Virtex II Partial Reconfiguration**.

3.3. Incorporación de Virtex II Partial Reconfiguration en Nessy

Una vez que la herramienta **Virtex II Partial Reconfiguration** estuvo finalizada, se creó una nueva funcionalidad en Nessy 2.0 para poder utilizarla y poder insertar bitstreams parciales con bitflips y realizar un estudio de la robustez del circuito. Igualmente que en versiones anteriores de Nessy, se cargaba un fichero en la FPGA y se ejecutaba, creando el fichero Golden en base a la primera ejecución y comparando las posteriores ejecuciones. Usando la funcionabilidad nueva de reconfiguración parcial, Nessy empezaba a ejecutar un bucle en el que se integraba Virtex II Partial Reconfiguration para modificar solamente un bit de una palabra de una frame ambos determinados por Nessy, y luego comparar la ejecución con el Golden. Si detectaba algún error, lo guardaba en un fichero para su posterior estudio y restauraba la frame. Luego volvería al inicio del bucle y así hasta que recorriese toda la FPGA.

Como esto es muy trabajoso debido al gran número de frames y bits, en total 11.575.552 bits de configuración, se decidió acotar el trabajo utilizando restricciones para la localización del diseño en la FPGA.

Para acotar la localización de un diseño en la FPGA hay que saber el número de slices que posee por columna y por filas. En este caso con la Virtex-II Pro tiene por columna slices que van del x0y0 al x0y159, mientras que por fila posee slices del x0y0 al x91y0. Por lo que se procedió a acotar el rango de slices configurables, mediante la inserción de estas restricciones en el archivo .ucf del proyecto VHDL.

Las instrucciones que se necesarias para realizar esto fueron:

INST “Cont_Tx_Serie” AREA_GROUP = AG_Contador;

AREA_GROUP “AG_Contador” RANGE = SLICE_X0Y0:SLICE_X1Y109;

AREA_GROUP “AG_Contador” MODE = RECONFIG;

Con estas instrucciones se consigue restringir el rango de slices configurables desde el slice x0y0 hasta el slice x1y109. Gracias a esto se pudo trabajar mejor en el estudio de los errores que se podrían presentar en el circuito.

4. Virtex II Configuration Viewer

4.1. Objetivo

El objetivo de la aplicación Virtex II Configuration Viewer es la de leer cualquier bitstream compatible con una Virtex-II Pro mediante un archivo de bytes (.bit), y crear un informe explicando claramente qué configura y cuáles son los comandos que componen el bitstream. La utilidad principal de esta herramienta es la de comprender mejor la estructura de un mapa de bits para luego poder crear o modificar cualquier bitstream sin producir ningún error.

4.2. Diseño

Virtex II Configuration Viewer es una aplicación diseñada en Java y que se ejecuta mediante consola. Para ejecutar la aplicación por consola, se le introduce por parámetro la ruta de dirección y el nombre de un archivo ".bit". Si en dicho archivo existe un bitstream de configuración completo o parcial compatible con una Virtex-II Pro, lo lee y crea otro archivo de texto con el mismo nombre que el archivo introducido por parámetro y como extensión ".v2cv". Este archivo que se crea es el informe que indica las propiedades del bitstream. Aunque tenga distinta extensión que un típico archivo de texto (txt), se puede abrir con cualquier editor de texto.

Un ejemplo de funcionamiento sería introducir por consola:

C:\Users\Admin>java -jar Virtex_II_Configuration_Viewer ./Examples/contador.bit

Como resultado se obtiene un archivo contador.bit.v2cv que presenta en cada línea de forma ordenada los comandos del bitstream. Como estos comandos están guardados en el archivo como cadenas de bytes, las palabras de comandos aparecen en hexadecimal y en binario. Una descripción de la funcionalidad del comando aparece en la línea de abajo.

Como resultado con los datos del ejemplo del bitstream del archivo contador.bit se obtiene:

AA995566 - 10101010100110010101010101100110
Sync Word – La palabra de sincronización sirve para alinear las palabras del bitstream

30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD – Este es el comando para escribir una palabra en el registro CMD
00000007 - 00000000000000000000000000000111
Packet Data: Word 1 CMD Register = RCRC Command – La palabra que se escribe en el registro CMD, indicando que se trata de un comando RCRC y que sirve para resetear el registro del CRC

30016001 - 00110000000000010110000000000001
Type 1 Write 1 Words to FLR – El comando para escribir una palabra en el registro FLR
000000CD - 00000000000000000000000011001101
Packet Data: Word 1 Frame Length Register = 205 Words = 0x000000CD – Se escribe en el registro FLR el tamaño de la longitud de la frame, esta es de 205 palabras.

30012001 - 00110000000000010010000000000001
Type 1 Write 1 Words to COR – Comando para escribir una palabra en el registro COR
00053FE5 - 00000000000001010011111111100101
Packet Data: Word 1 Configuration Options Register = 0x00053FE5 – Palabra que configura distintas opciones de configuración y se almacenan en el registro COR
3001C001 - 00110000000000011100000000000001
Type 1 Write 1 Words to IDCODE – Comando para escribir en el registro IDCODE
0127E093 - 00000001001001111110000010010011
Packet Data: Word 1 Device ID = 0x0127E093 (XC2VP30) – Palabra que indica que el ID del dispositivo es el XC2VP30

3000C001 - 00110000000000001100000000000001
Type 1 Write 1 Words to MASK – Comando para escribir una palabra en el registro MASK
00000000 - 00000000000000000000000000000000
Packet Data: Word 1 Mask Register = 0x00000000 – Palabra que indica la máscara que se utilizará almacenandola en el registro MASK

30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD
00000009 - 00000000000000000000000000001001
Packet Data: Word 1 CMD Register = SWITCH Command – Comando para actualizar la frecuencia del reloj del dispositivo

30002001 - 00110000000000000010000000000001
Type 1 Write 1 Words to FAR – Comando para escribir una palabra en el registro FAR
00000000 - 00000000000000000000000000000000
Packet Data: Word 1 Frame Address Register = 0x00000000 – Esta palabra indica la dirección por la que se empezará a configurar las frames y se almacena en el registro FAR

30004000 - 00110000000000000100000000000000
Type 1 Write 0 Words to FDRI – Comando para indicar que se va a escribir en el registro FDRI
500585D6 - 01010000000001011000010111010110
Type 2 Write 361942 Words to FDRI – Comando de tipo 2 para indicar que se va a escribir un gran numero de palabras en el registro FDRI
00000000 - 00000000000000000000000000000000
Packet Data: Word 1 Frame Data Input Register = 0x00000000
00000000 - 00000000000000000000000000000000
Packet Data: Word 2 Frame Data Input Register = 0x00000000
“...”
C2010002 - 11000010000000010000000000000010
Packet Data: Word 198497 Frame Data Input Register = 0xC2010002
40001000 - 01000000000000000001000000000000
Packet Data: Word 198498 Frame Data Input Register = 0x40001000

60814140 - 01100000100000010100000101000000
Packet Data: Word 198499 Frame Data Input Register = 0x60814140
“...”
00000000 - 00000000000000000000000000000000
Packet Data: Word 361941 Frame Data Input Register = 0x00000000
00000000 - 00000000000000000000000000000000
Packet Data: Word 361942 Frame Data Input Register = 0x00000000

00007191 - 00000000000000000111000110010001
Packet Data: AutoCRC Word = 0x00007191 – La ultima palabra del paquete de configuración del FDRI se destina a una comprobación de CRC

30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD
0000000A - 00000000000000000000000000001010
Packet Data: Word 1 CMD Register = GRESTORE Command – Este comando sirve para resetear los IOBs y flipflops de los CLBs a sus valores inciales

30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD
00000003 - 00000000000000000000000000000011
Packet Data: Word 1 CMD Register = LFRM Command – Comando para indicar la escritura de la ultima frame

20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 1 – Comando de intrucción NOOP que sirve para liberar los buffers
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 2
“...”
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 206

30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD
00000005 - 00000000000000000000000000000101
Packet Data: Word 1 CMD Register = START Command – Comando para iniciar la secuencia de startup de la FPGA

3000A001 - 00110000000000001010000000000001
Type 1 Write 1 Words to CTL – Este comando sirve para escribir una palabra en el registro CTL
00000000 - 00000000000000000000000000000000
Packet Data: Word 1 Control Register = 0x00000000 – Con esta palabra se indican las distintas opciones de seguridad de la FPGA almacenándola en el registro CTL

30000001 - 00110000000000000000000000000001
Type 1 Write 1 Words to CRC
00005F57 - 00000000000000000101111101010111
Packet Data: Word 1 CRC Register = 0x00005F57 – Comprobación de CRC

30008001 - 00110000000000001000000000000001
Type 1 Write 1 Words to CMD
0000000D - 00000000000000000000000000001101
Packet Data: Word 1 CMD Register = DESYNCH Command – Este comando sirve para desincronizarse del dispositivo al final de la configuración

20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 1
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 2
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 3
20000000 - 00100000000000000000000000000000
Type 1 NOOP Word 4

La estructura de Virtex II Configuration Viewer es sencilla. Primero se lee el archivo introducido por parámetro, luego se clasifican los distintos comandos del bitstream para luego crear y almacenar el resultado de cada comando en el archivo de salida.

El bucle que clasifica los distintos comandos es el que posee mayor interés, en él se comprueba si es una palabra de sincronización comparándola con la palabra de sincronización “AA995566”, si es una NOOP “20000000” y sino se evalúa la cabecera, mediante el tipo de palabra usando la función evaluatePacket(). También se evalúa si es de lectura o escritura mediante evaluateWR() y el número de palabras que la compone con evaluateTypeWord(). Después se evalúa el paquete de datos dependiendo del comando de la cabecera.

5. Virtex II Configuration Comparer

5.1. Objetivo

El objetivo de la aplicación Virtex II Configuration Comparer es la de leer dos bitstream para una Virtex-II Pro mediante dos archivos de bytes (.bit), y crear un informe de cada uno explicando claramente qué configura y cuáles son los comandos que componen cada bitstream correspondiente. La utilidad principal de esta herramienta es la de comparar las diferencias entre dos mapas de bits para luego poder detectar errores en los mapas de configuración. Como resultado si en un sistema real, durante la ejecución cae una partícula, se podría hacer un readback del mapa de bits y comparar con el bitstream original y el que se ha leído. Se podría saber dónde se habría producido el SEU y restaurar el circuito al circuito original. Esto se asemeja a sistemas de recuperación de errores existentes que salvan periódicamente el estado marcándolo como un checkpoint cada cierto tiempo, y son capaces de recuperarse desde el último checkpoint realizado si se ha producido un fallo.

5.2. Diseño

Virtex II Configuration Comparer es una aplicación diseñada en Java y que se ejecuta mediante consola. Se introduce por parámetro la ruta de dirección y el nombre de dos archivos ".bit". Si se trata de dos bitstreams completos compatibles con una Virtex-II Pro, los lee y crea tres archivos de texto. Dos de estos archivos son los nombres de los archivos introducidos acabados en ".v2cv", es decir se crean dos archivos iguales a como los crea la herramienta Virtex II Configuration Viewer, con la excepción de que además en cada comando introduce un índice para enumerarlos. Esta enumeración sirve para el otro archivo que refleja las diferencias entre los dos, indicando el índice donde se encuentra la diferencia entre los dos mapas de bits y así de una rápida localización. Este archivo se crea concatenando el nombre de los dos archivos y como extensión "_Comparision.v2cc". Aunque tenga distinta extensión que un típico archivo de texto (txt), se puede abrir con cualquier editor de texto.

Un ejemplo de funcionamiento sería introducir por consola:

C:\Users\Admin>java -jar Virtex_II_Configuration_Comparer ./Examples/contador.bit ./Examples/contadorModificado.bit

Como resultado se obtienen los archivos contador.bit.v2cv y contadorModificado.bit.v2cv y un archivo contador.bit_contadorModificado.bit_Comparision.v2cc con el siguiente contenido:

En este informe se indica primeramente en qué índice de línea se encuentra la diferencia. En una línea posterior indica el nombre del archivo y cual es la diferencia que se ha encontrado, así se van presentando todas las diferencias entre los dos archivos.

Como resultado con los datos del ejemplo se obtiene:

Diference at Line : 138025
contador.bit : 5F08FFFF - 01011111000010001111111111111111
Packet Data: Word 138007 Frame Data Input Register = 0x5F08FFFF
contadorModificado.bit : 5F00FFFF - 01011111000000001111111111111111
Packet Data: Word 138007 Frame Data Input Register = 0x5F00FFFF

Diference at Line : 361961
contador.bit : 00007191 - 00000000000000000111000110010001
Packet Data: AutoCRC Word = 0x00007191
contadorModificado.bit : 0000A4B3 - 00000000000000001010010010110011
Packet Data: AutoCRC Word = 0x0000A4B3

Como se observa se diferencian en un comando de escritura en el FDRI, en el cual solo hay modificado un bit y en el AutoCRC que al tener ese bit modificado del comando FDRI los CRC son totalmente distintos.

La estructura Virtex II Configuration Comparer es una extensión de la herramienta Virtex II Configuration Viewer, pero ampliada para que compare los dos tipos de bitstreams y muestre las diferencias entre ambos.

6. Virtex II Partial Reconfiguration

6.1. Objetivo

El objetivo de la herramienta Virtex II Partial Reconfiguration es la de leer cualquier bitstream compatible con una Virtex-II Pro mediante un archivo de bytes (.bit), y crear otros dos bitstreams de configuración parciales. Uno de ellos sirve para introducir un bitflip en el mapa de bits original, para simular un SEU en la posición que se le indique, y el otro servirá para restaurar el bitflip que pueda producir el bitstream anterior. La utilidad principal de esta herramienta es la de introducir bitflips en una FPGA reconfigurándola parcialmente como si de un SEU se tratara. Una vez se hayan obtenido los errores que se producen durante la ejecución, se utiliza el otro bitstream para restaurar la FPGA a su estado anterior sin errores.

6.2. Diseño

Virtex II Partial Reconfiguration es una aplicación diseñada en Java y que se ejecuta mediante consola y a la que se le introduce por parámetro la ruta de dirección y el nombre de un archivo ".bit", el número de la palabra de frame que se quiere modificar y el bit que se quiere modificar. Si en dicho archivo existen un bitstream completo compatible con una Virtex-II Pro, lo lee y crea dos archivos ".bit". Uno de estos archivos es un bitstream de reconfiguración parcial capaz de insertar un bitflip en la posición de la frame que se ha indicado mediante parámetro y el otro es el bitstream de reconfiguración parcial capaz de restaurar el circuito original.

Un ejemplo de funcionamiento sería introducir por consola:

C:\Users\Admin>java -jar Virtex_II_Partial_Reconfiguration -i ./contador.bit
-o contadorModif.bit -f <número de la palabra de frame> -b <número de bit>

Como resultado se obtienen dos archivos contadorModif.bit y contadorModifRestorer.bit capaces de realizar la reconfiguración parcial de la frame indicada modificando o restaurando el bit indicado.

La estructura Virtex II Partial Reconfiguration es una extensión de la herramienta Virtex II Configuration Viewer, pero ampliada para que cree dos archivos bitstream para reconfiguración parcial de una sola frame.

Para el desarrollo de esta aplicación hubo que prestar atención a los problemas que se presentan a la hora de modificar manualmente un bit en el mapa de bits. Para realizar una modificación de un bitflip había que desactivar la comprobación de CRC en el comando COR. En el bit 29 del registro COR había que poner su valor a “1” para deshabilitar la comprobación de CRC. Además, como se explicó previamente, en los bitstreams parciales, hay que eliminar los comandos de GRESTORE que sirve para la reinicialización de la FPGA y de START que inician la ejecución de la FPGA.

7. Conclusión

Con la ayuda de Nessy 2.0, FPGA Editor y la implementación de Virtex II Configuration Viewer, Configuration Comparer y Partial Reconfiguration, se ha conseguido realizar una herramienta para insertar errores en las FPGAs de tipo Virtex-II Pro mediante la inserción de bitflips en mapas de configuración. Se había trabajado antes con archivos de configuración de FPGA a lo largo de la carrera, pero no a este nivel. Estos archivos eran como cajas negras, funcionaban pero no se sabía cómo. Gracias a este estudio se ha conseguido comprender cómo funciona un bitstream y profundizar en el conocimiento de su estructura y cómo se configuran las FPGAs. Aunque la configuración de un bitstream parece compleja, finalmente resultó fácil manejarla, no solo utilizando la reconfiguración parcial, sino introduciendo otro tipo de comandos a modo de estudio o investigación, como por ejemplo la anulación de CRC.

También se ha conseguido profundizar en el conocimiento de la arquitectura interna de la FPGA tipo Virtex-II Pro en lo que se refiere a sus celdas lógicas, slices, LUT, flipflops...

Donde resultaron el mayor número de complicaciones fue al utilizar java para programar las tres herramientas descritas, las funciones de lectura de bytes de un archivo no presentaban el efecto deseado al leer bytes de un archivo bitstream. Por lo que hubo que recurrir a funciones de bajo nivel y trabajar a muy bajo nivel para el simple hecho de leer un byte.

8. Trabajo Futuro

Dado el resultado obtenido con la FPGA de tipo Virtex-II Pro, como trabajo futuro se piensa realizar una ampliación en la plataforma de trabajo, es decir, llevar este proyecto a una FPGA de tipo Virtex-IV o Virtex-V. Además se pretende introducir en memoria el Test Bench y el Golden del circuito diseñado y configurado en la FPGA. Esto permitirá no tener que mandar y leer los datos por el puerto serie en cada ejecución y comparar con la ejecución Golden, como teníamos que hacer con la herramienta Nessy. Finalmente probar distintos métodos de detección y corrección de errores y así comprobar su validez con este entorno.

8. Bibliografía

1. Virtex-II Pro and Virtex-II pro X FPGA User Guide
   UG012 (v4.2) 5 November 2007 www.xilinx.com
2. Application Note: Virtex, Virtex-E, Virtex-II, Virtex-II Pro Families
   Two Flows for Partial Reconfiguration: Module Based of Difference Based
   XAPP290 (v1.2) September 9, 2004 www.xilinx.com
3. Application Note: Virtex Architectures Difference-Based Partial Reconfiguration
   XAPP290 (v2.0) December 3, 2007 Author: Emi Eto www.xilinx.com
4. AREA_GROUP Virtex
   www.xilinx.com
5. "Inserción y detección de errores en dispositivo FPGA"
   Autores: Carlos Sánchez-Vellisco, Antonio José García y David Fernández
6. "Técnicas de inyección de fallos basadas en FPGAs para la evaluación de la tolerancia a fallos de tipo SEU en circuitos digitales" Universidad Carlos III de Madrid 2007
   Autora: Marta Portela García
7. The use of reprogrammable FPGAs in the space
   http://www.esa.int/TEC/Microelectronics/
8. FPGAs in Space
   http://www.brianhpratt.net
9. Circuitos Integrados configurables ASIC
   http://icprgm-asic.blogspot.com/2007_12_01_archive.html

Apéndice 1: Plataforma de Trabajo

Placa XUP Virtex-II Pro Development System

La placa que se utilizó para realizar este trabajo y todas las investigaciones, fue una XUP basada en una Virtex-II Pro. Esta tarjeta posee una memoria SDRAM de hasta 2GBytes, distintos puertos de entrada/salida, como por ejemplo SATA, conexión a Ethernet, audio, USB2, puerto PS/2, puerto serie RS-232. También posee 5 botones, 4 switches y 4 LEDs que se pueden configurar para poder interactuar con la FPGA. En la siguiente imagen se muestra su estructura (Figura Ap1.1).

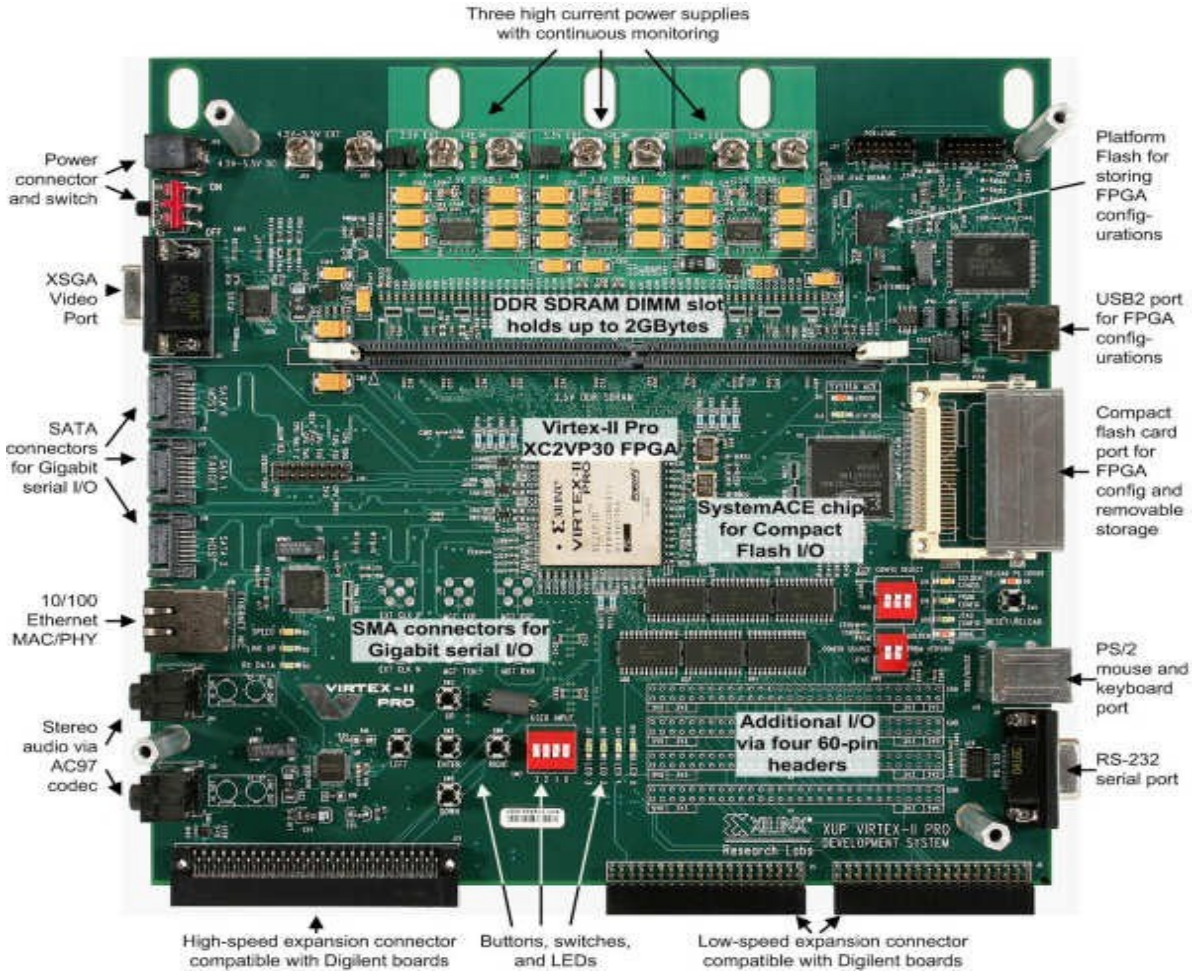

Figura Ap1.1: Placa XUP Virtex-II Pro Development System

Arquitectura de las FPGAs tipo Virtex

Las FPGA de tipo Virtex poseen al menos 3 bloques bien diferenciados

1. El Configurable-Logic Blocks (CLB), es el bloque donde se implementan los circuitos lógicos.
2. Input-Output Block (IOB), lugar en el que se conectan las configuraciones internas con pines de entrada/salida.
3. Los Digital Clock Managers (DCM), permiten gestionar las señales del reloj a toda la placa.

La arquitectura básica de FPGA de tipo Virtex sería como se muestra en la siguiente imagen (Figura Ap1.2).

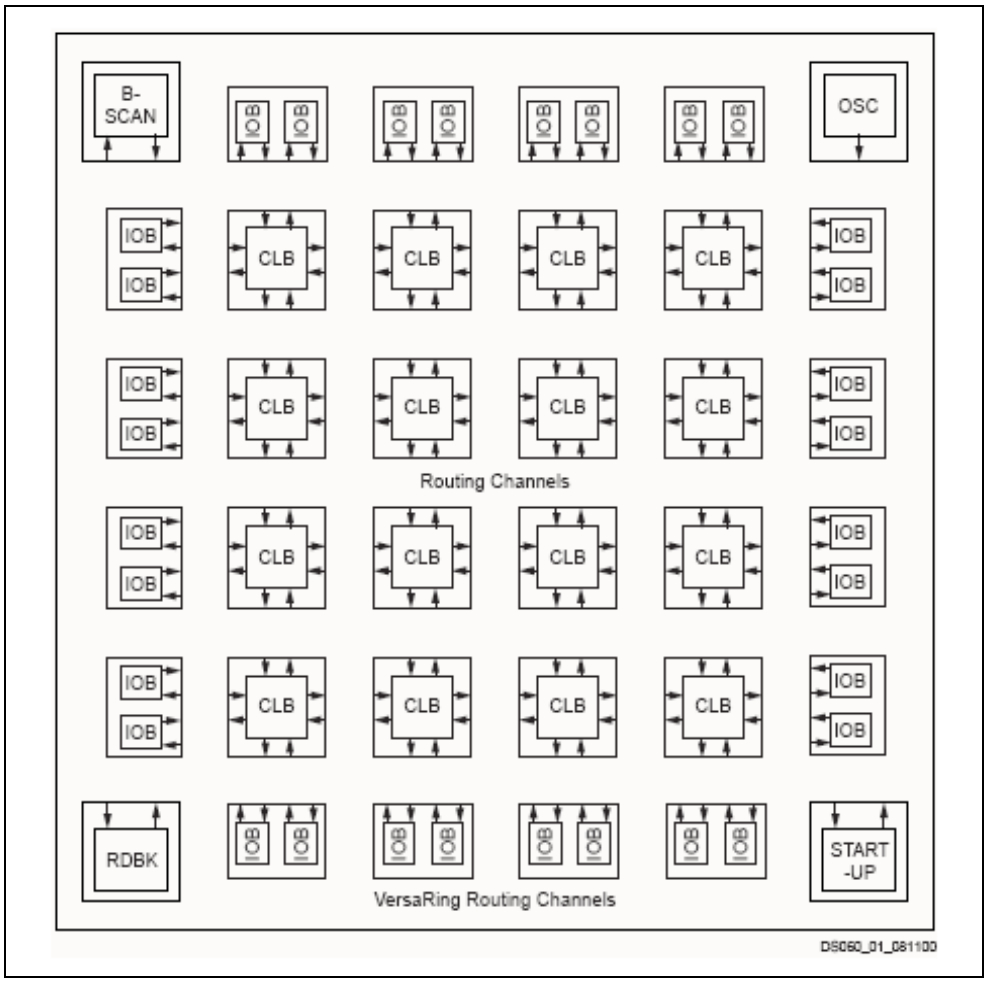

Figura Ap1.2: Arquitectura de una FPGA del tipo Virtex

El CLB es el elemento esencial de una FPGA. Como se ha explicado antes es el bloque donde se implementan los circuitos lógicos, pueden implementarse tanto circuitos combinacionales como secuenciales. En la Virtex-II Pro cada CLB poseen cuatro celdas lógicas distribuidas en dos slices. En la siguiente imagen se muestran las celdas lógicas de un CLB (Figura Ap1.3).

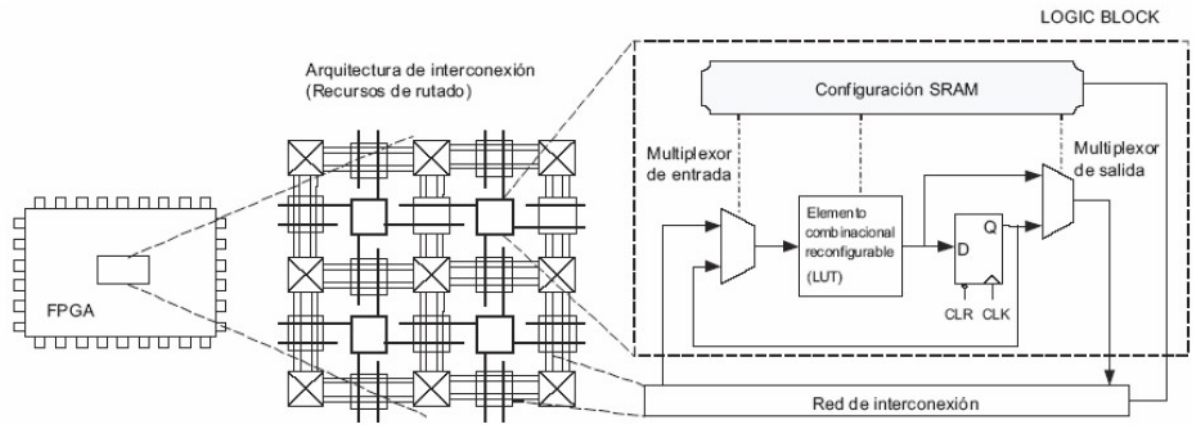

Figura Ap1.3: Celdas Lógicas de un CLB

Como se puede observar en la imagen, un elemento de la celda lógica es la LookUp Table (LUT). La LUT es una memoria RAM en la que se almacenan valores, y es uno de los elementos en el que se centrará el estudio de este trabajo. Esto es debido a que es el elemento combinacional reconfigurable y en la que se pueden presentar errores debido a su vulnerabilidad frente a los SEUs.

Configuración de una FPGA del tipo Virtex II

La configuración de una FPGA tipo Virtex II se realiza mediante frames verticales de un bit de ancho y de largo igual a la altura del dispositivo. Estas frames son la mínima unidad de configuración direccionable, es decir, la configuración de una FPGA solo se puede realizar indicando la dirección de la frame que se quiere configurar, para después configurarla entera de arriba a abajo.

El número de frames necesarias para configurar una FPGA varia según el tipo de dispositivo. En el caso de la Virtex-II Pro que se utilizó en este proyecto (XC2VP30), necesita 1.756 frames de configuración y cada frame tiene una longitud de 206 palabras de 32 bit cada una. Esto hace un total de 11.575.552 bits de configuración. El cálculo es trivial:

$$1756\ frames * (206\ palabras * 32\ bits) = 11575552\ bits$$

Tipos de Frames de Configuración

Las frames de configuración se agrupan en seis tipos de columnas, cada una para configurar los dispositivos IOB, IOI, CLB, GCLK, BlockRAM y la interconexión de los BlockRAM.

1. El IOB configura el voltaje estándar de las entradas/salidas del dispositivo. La FPGA que se utilizó en este proyecto posee 2 columnas de IOBs de 4 frames por columna.
2. El IOI configura los registros, multiplexores y buffers de los IOBs de la derecha e izquierda. En este caso hay 2 columnas de IOIs de 22 frames cada columna en la Virtex-II Pro.
3. El CLB programa la configuración de los bloques lógicos, enrutado e interconexión, así como programa los IOBs de la parte de arriba y de abajo del dispositivo. El número de columnas de CLBs es de 46 y 22 frames por columna.
4. Las columnas BlockRAM solamente configuran la BlockRAM del espacio de memoria del usuario. Son 8 columnas en el dispositivo de 64 frames por columna.
5. Las interconexiones y el resto de aspectos de la BlockRAM se configuran mediante la columna BlockRAM Interconnect. Tiene un tamaño de 8 columnas y 22 frames por columna para el dispositivo que se utilizó en este proyecto.
6. Por último la columna del GCLK configura los dispositivos asociados al reloj global, incluyendo sus buffers y DCMs. En las Virtex-II Pro solo se necesita 1 columna de 4 frames para configurarla.

Paquetes del Bitstream

Para configurar una FPGA se carga en esta un bitstream o mapa de bits. Este bitstream está compuesto de una palabra de sincronización de 32 bits (0xAA995566) y numerosos paquetes de datos. La palabra de sincronización sirve para alinear los dispositivos lógicos de configuración de la FPGA con el primer paquete de datos del bitstream. Los paquetes de datos del bitstream consisten en una cabecera de 32 bits y un cuerpo de longitud variable. Existen dos tipos de paquetes:

1. El tipo 1, utilizado para paquetes pequeños, hasta $2^{11}-1$ palabras (Figura Ap1.4).
2. El tipo 2, utilizado para paquetes largos, hasta $2^{27}-1$ palabras (Figura Ap1.5).

Type			W R	R D	Register Address														RSVD		Word Count										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x

Figura Ap1.4: Cabecera de paquete Tipo 1

Type			W R	R D	Word Count																										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura Ap1.5: Cabecera de paquete Tipo 2

En la cabecera se indica qué comando se quiere realizar, ya sea escribir o leer y en qué tipo de registro del dispositivo. También se pueden encontrar comandos que no configuran nada y que únicamente sirven para liberar los buffers del último comando que se ha procesado, son las NOOPs (0x20000000).

Solo ciertos registros del dispositivo se pueden leer y escribir, otros solo se pueden leer o escribir. Cada registro está identificado por una dirección para poder configurarlos (figura Ap1.6).

1. El registro Cycle Redundancy Checks (CRC) almacena una palabra de 16 bits de CRC. Este registro se puede tanto escribir como leer. Se puede acceder a el directamente indicándolo en un comando o en un paquete FDRI la ultima palabra es para escribir un CRC automático.

   Así con esta palabra de CRC se comprueba la consistencia de los paquetes FDRI que se envían a la FPGA.

2. El registro Frame Address Register (FAR) se configura para indicar cual es la dirección de la frame que se quiere configurar.

3. El registro Frame Data Input Register (FDRI) sirve para cargar las frames de configuración. Cada vez que se llena su contenido, se vuelca la frame en memoria en la dirección indicada por el registro FAR, y ésta se incrementa automáticamente en 1.

Register Name	Read	Write	Address	Description
CRC	Y	Y	00000	CRC register
FAR	Y	Y	00001	Frame address register
FDRI	N	Y	00010	Frame data input register (write configuration data)
FDRO	Y	N	00011	Frame data output register (readback configuration data)
CMD	Y	Y	00100	Command register
CTL	Y	Y	00101	Control register
MASK	Y	Y	00110	Masking register for CTL
STAT	Y	N	00111	Status register
LOUT	N	Y	01000	Legacy output register (DOUT for daisy chain)
COR	Y	Y	01001	Configuration option register
MFWR	N	Y	01010	Multiple frame write register
FLR	Y	Y	01011	Frame length register
KEY	N	Y	01100	Initial key address register
CBC	N	Y	01101	Initial CBC value register
IDCODE	Y	Y	01110	Device ID register

Figura Ap1.6: Tabla de direcciones de registros configurables

4. Para poder leer los valores en memoria de las frames configuradas mediante los registros FDRI, hay que acceder al registro de solo lectura Frame Data Output Register (FDRO). Al igual que FDRI empezará a leer las direcciones de memoria empezando por la indicada en el FAR.

5. Existen comando de control para la configuración de los frames. Estos comandos se indican mediante el Command Register (CMD). Hay un gran numero de comandos distintos y se identifican con su código de comando (Figura Ap1.7).

6. Para configurar el nivel de seguridad se utiliza el Control Register (CTL) (Figura Ap1.8). Por ejemplo en este registro se puede indicar la persistencia de la FPGA, que será útil cuando se necesite reconfigurar la FPGA mientras esta ejecutándose. Los niveles de seguridad se configuran modificando los últimos bit del registro (Figura Ap1.9).

Command	Code	Description
WCFG	0001	Write Configuration Data. Used prior to writing configuration data to the FDRI.
MFWR	0010	Multiple Frame Write Register. Used to perform a write of a single frame to multiple frame addresses.
LFRM	0011	Last Frame. Deasserts the GHIGH_B signal, activating all interconnect. The GHIGH_B signal is asserted with the AGHIGH command.
RCFG	0100	Read Configuration Data. Used prior to reading configuration data from the FDRO.
START	0101	Begin Startup Sequence. Initiates the startup sequence. The startup sequence begins after a successful CRC check and a DESYNCH command are performed.
RCAP	0110	Reset Capture. Resets the CAPTURE signal after performing readback-capture in single-shot mode.
RCRC	0111	Reset CRC. Resets the CRC register
AGHIGH	1000	Assert GHIGH_B Signal. Places all interconnect in a high-Z state to prevent contention when writing new configuration data. This command is only used during shutdown reconfiguration and readback. Interconnect is reactivated with the LFRM command.
SWITCH	1001	Switch CCLK Frequency. Updates the frequency of the Master CCLK to the value specified by the OSCFSEL bits in the COR.
GRESTORE	1010	Pulse the GRESTORE Signal. Sets/resets (depending on user configuration) IOB and CLB flip-flops. See "Readback Capture," page 382.
SHUTDOWN	1011	Begin Shutdown Sequence. Initiates the shutdown sequence, disabling the device when finished. Shutdown activates on the next successful CRC check or RCRC instruction (typically an RCRC instruction).
GCAPTURE	1100	Pulse GCAPTURE. Loads the capture cells with the current register states.
DESYNCH	1101	Reset DALIGN Signal. Used at the end of configuration to desynchronize the device. After desynchronization, all values on the configuration data pins are ignored.

Figura Ap1.7: Código de los comandos del CMD

																								Reserved	SBITS		PERSIST	Reserved	Reserved	GTS_USR_B
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	0	0	x

Figura Ap1.8: Registro CTL

Name	Bit Index	Description
Reserved	Various	Reserved CTL register bits for testing. Always leave these bits set to 0.
SBITS	5:4	Security Level: 0: Read/Write OK (default) 1: Readback disabled 2,3: Readback disabled, writing disabled except to the CRC register.
PERSIST	3	The configuration interface defined by M2:M0 remains after configuration. Typically used only with the SelectMAP interface to allow reconfiguration and readback. 0: No (default) 1: Yes
GTS_USER_B	0	Active Low signal to place I/Os into a high-Z state. 0: I/Os placed in high-impedance state 1: I/Os active

Figura Ap1.9: Bits de configuración del CTL

7. Mask Register (MASK) es el registro que se encarga de hacer la mascara de los bits del CTL para permitir modificarlos.

8. Un registro de solo lectura es el Status Register (STAT) (Figura Ap1.10). Este registro indica el estado de las señales globales de la FPGA, sus valores pueden ser leídos mediante las interfaces SelectMAP o JTAG (Figura Ap1.11.1 y Ap1.11.2).

																BAD_KEY_SEQ	DEC_ERROR	ID_ERROR	DONE	INIT	MODE			GHIGH_B	GWE	GTS_CFG_B	IN_ERROR	DCI_MATCH	DCM_LOCK	PART_SECURED	CRC_ERROR
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura Ap1.10: Registro STAT

9. El Legacy Output Register (LOUT) se usa para escribir datos en el pin DOUT, sin embargo no esta permitido utilizarlo con modos de configuración mediante SelectMAP y JTAG.

Name	Bit Index	Description
BAD_KEY_SEQ	15	Encryption keys are stored with a tag indicating their position among the other keys. If the keys are sent in the wrong order, it will be indicated by this status bit. 0: No decryptor key sequence error 1: Devryptor keys were not used in the correct sequence
DEC_ERROR	14	Write FDRI issued before or after decrypt operation. 0: No decryptor error 1: Decryptor error. Assert PROG to recover
ID_ERROR	13	Attempt to write to FDRI without successful IDCODE check. 0: No ID_ERROR 1: ID_ERROR. Assert PROG to recover
DONE	12	The DONE bit in the STATUS register reflects the actual level on the DONE pin. If the DONE signal is released by the device but held Low externally, this bit remains Low. 0: DONE pin = logic 0 1: DONE pin = logic 1
INIT	11	The INIT bit in the STATUS register reflects the actual level on the INIT pin. 0: INIT pin = logic 0 1: INIT pin = logic 1
MODE	10:8	Status of the MODE pins (M2:M0).
GHIGH_B	7	Status of the GHIGH_B signal. 0: GHIGH_B is asserted 1: GHIGH_B is deasserted
GWE	6	Status of the Global Write Enable signal. 0: All FFs and BRAMs are write disabled 1: FFs and BRAMs are write enabled
GTS_CFG_B	5	Status of the GTS_CFG_B signal. 0: All I/Os in high-Z state 1: I/Os are not placed in High-Z by this signal Note that there are other ways of forcing I/Os into a high-Z state, including GTS_USR_B, as well as through various JTAG instructions.
IN_ERROR	4	Indicates that input data is being clocked in too fast. This can happen if the CCLK frequency exceeds $F_{CC\_SELECTMAP}$ in SelectMAP mode. 0: No legacy input error 1: Legacy input error

Figura Ap1.11.1: Valores del registro STAT

Name	Bit Index	Description
DCI_MATCH	3	This signal reflects the logical AND of all MATCH signals. There is one MATCH signal per I/O bank. If no DCI I/Os are instantiated in a given bank,the bank's MATCH signal = 1. 0: DCI not matched 1: DCI is matched
DCM_LOCK	2	This is a logical AND function of all DCM LOCKED signals. Unused DCM LOCKED signals =1. 0: DCMs not locked 1: DCMs are locked
PART_SECURED	1	Status of the Triple-DES decryptor. 0: Decryptor security not set 1: Decryptor security set
CRC_ERROR	0	Status of the CRC. 0: No CRC error 1: CRC error

Figura Ap1.11.2 : Valores del registro STAT (Continuación)

10. Para determinar ciertas configuraciones del dispositivo, como la revisión del CRC, se hace mediante el Configuration Options Register (COR) (Figura Ap1.12). Existen muchos tipos de configuración (Figura Ap1.13.1 y Ap1.13.2).

RESVRED	RESVRED	CRC_BYPASS	RESERVED		SHUT_RST_DCM	DONE_PIPE	DRIVE_DONE	SINGLE	OSCFSEL						SSCLKSRC		DONE_CYCLE			MATCH_CYCLE			LOCK_CYCLE			GTS_CYCLE			GWE_CYCLE		
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figura Ap1.12: Registro COR

11. Si se desea utilizar o la opción de compresión de bitstream está activa en el Bitgen, mediante el registro Multiple Frame Write Register (MFWR) se puede indicar que varias frames de datos son identicas y así se hace copia de la misma frame en múltiples direcciones de la memoria.

12. El registro Frame Length Register (FLR) almacena la longitud de frame del dispositivo, esta debe indicarse antes de configurar alguna operación de FDRI o FDRO.

13. Initial Key Address Register (KEY), este registro de solo escritura, indica el número de DES keys, que es utilizado para la encriptación del bitstream.

14. Cipher Block Chaining Register (CBC), también de solo escritura, almacena el tipo de encriptación DES keys.

Name	Bit Index	Description
CRC_BYPASS	29	0: CRC enabled 1: CRC disabled
SHUT_RST_DCM	26	When this bit is set, all DCMs will be reset if the AGHIGH command is issued. 0: DCMs cannot be reset through the configuration logic 1: DCMs will reset during shutdown readback or reconfiguration
DONE_PIPE	25	Adds a pipeline stage for the DONEIN signal. The DONEIN signal reflects the logic level on the DONE pin, not whether the device has released the DONE pin. This pipeline stage can be useful if the rise time on the DONE pin is slow, which could otherwise cause configuration to fail. 0: No pipeline stage for DONEIN 1: Add pipeline stage for DONEIN
DRIVE_DONE	24	By default the DONE pin is an open-drain output, meaning that the pin actively drives a logic '0' before configuration, then goes into a high-Z state when it is "released" during Startup. This setting requires an external pull-up resistor on the DONE pin, and is typically applied to Slave devices in a serial daisy-chain. Alternately, the DONE pin can be configured as an actively-driven output, meaning that the pin actively drives a logic '1' when it is released. 0: DONE pin is an open-drain output (released = high-Z) 1: DONE pin is an actively driven output (released = logic '1')
SINGLE	23	The SINGLE bit determines whether the CAPTURE_VIRTEX2 primitive operates in one-shot mode or in continuous sample mode. This bit is set when the "one-shot" attribute is attached to the CAPTURE_VIRTEX2 primitive (see "Readback Capture," page 382). 0: SINGLE mode disabled. In this mode, capture values are loaded on every clock cycle while CAP is asserted. 1: SINGLE mode enabled. In this mode, capture values are loaded only once, regardless of how many clock cycles the CAP signal is asserted for.
OSCFSEL	22:17	Selects the CCLK frequency for Master configuration modes (see Table 4-29).
SSCLKSRC	16:15	Startup sequence clock source: 00: CCLK 01: UserClk (per connection on the CAPTURE_VIRTEX2 block) 1x: JTAGClk

Figura Ap1.13.1: Valores del registro COR

Name	Bit Index	Description
DONE_CYCLE	14:12	Startup cycle to release the DONE pin: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6
MATCH_CYCLE	11:9	Startup cycle to stall in until DCI matches: 000: Startup cycle 0 001: Startup cycle 1 010: Startup cycle 2 011: Startup cycle 3 100: Startup cycle 4 101: Startup cycle 5 110: Startup cycle 6 111: No Wait
LOCK_CYCLE	8:6	Startup cycle to stall in until DCMs lock: 000: Startup cycle 0 001: Startup cycle 1 010: Startup cycle 2 011: Startup cycle 3 100: Startup cycle 4 101: Startup cycle 5 110: Startup cycle 6 111: No Wait
GTS_CYCLE	5:3	Startup cycle to deassert the Global Three-State (GTS) signal: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6
GWE_CYCLE	2:0	Startup phase to deassert the Global Write Enable (GWE) signal: 000: Startup cycle 1 001: Startup cycle 2 010: Startup cycle 3 011: Startup cycle 4 100: Startup cycle 5 101: Startup cycle 6

Figura Ap1.13.2: Valores del registro COR (Continuación)

15. Existen distintos tipos de dispositivos de la Virtex-II Pro, para que no se produzcan errores al cargar el bitstream debido al tipo de dispositivo, existe una comprobación del identificador del dispositivo y el del bitstream indicado por Device ID (IDCODE) (Figura Ap1.14).

Device	IDCODE	Device	IDCODE
XC2VP2	0x01226093	XC2VP40	0x01292093
XC2VP4	0x0123E093	XC2VP50	0x0129E093
XC2VP7	0x0124A093	XC2VP70	0x012BA093
XC2VP20	0x01266093	XC2VPX70	0x018BA093
XC2VPX20	0x01866093	XC2VP100	0x012D6093
XC2VP30	0x0127E093		

Figura Ap1.14: Tipos de dispositivos Virtex-II Pro junto son su identificador

Apéndice 2: Códigos Fuente

Código Virtex II Configuration Viewer

Main.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package virtexiiconfigurationviewer;

import IO.*;
import Program.*;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Main
{

    /**
     * @param args the command line arguments
     */

    private final static int BEGINPACKETBITS = 31;
    private final static int ENDPACKETBITS = 29;
    private final static int WRITEBIT = 28;
    private final static int READBIT = 27;
    private final static int BEGINTYPE1WORD = 10;
    private final static int ENDTYPE1WORD = 0;
    private final static int BEGINTYPE2WORD = 26;
    private final static int ENDTYPE2WORD = 0;
    private final static int BEGINCONFIGREG = 17;
    private final static int ENDCONFIGREG = 13;
    private final static int BEGINCMD = 3;
    private final static int ENDCMD = 0;
    private final static String[] READWRITE = {" "," Read ", " Write "};
    private final static String[] CONFIGREG = {"CRC","FAR","FDRI","FDRO","CMD",
```

```
      "CTL","MASK","STAT","LOUT","COR","MFWR","FLR","KEY","CBC","IDCODE"};
   private final static String[] CMD = {" ","WCFG","MFWR","LFRM","RCFG","START",
      "RCAP","RCRC","AGHIGH","SWITCH","GRESTORE","SHUTDOWN","GCAPTURE","D
ESYNCH"};

   private static Read input;
   private static Write output;

   public static void main(String[] args)
      {
      // TODO code application logic here

      Main.input = new Read(args[0]);
      Main.output = new Write(args[0]);
      Program program = new Program();
      Main.input.readFile(program);
      Main.translate(program);
      Main.output.writeProgram(program);
      }

   private static void translate(Program program)
      {
      Instruccion instruccion;
      int index = 0;
      String configuration;
      while(index < program.getSize())
         {
         instruccion = program.getInstruccion(index);
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words ";
         if(instruccion.getInstruccionHex().compareTo("AA995566") == 0)
            {
            configuration = "Sync Word";
            }
         else if(instruccion.getInstruccionHex().compareTo("20000000") == 0)
            {
            int count = 1;
            while(instruccion.getInstruccionHex().compareTo("20000000") == 0)
               {
               configuration = "Type " + Main.evaluatePacket(instruccion) +
                     " NOOP Word " + count;
               program.addConfiguration(index,configuration);
               count++;
               index++;
               if(index < program.getSize())
                  instruccion = program.getInstruccion(index);
               else
                  return;
               }
```

```
   index--;
   }
else if(Main.evaluatePacket(instruccion) == 1)
   {
   if(Main.evaluateWR(instruccion) == 2)
      {
      if(Main.evaluateConfReg(instruccion) < Main.CONFIGREG.length)
         configuration += "to " + Main.CONFIGREG[Main.evaluateConfReg(instruccion)];
      else
         configuration += "to Not Configuration Register Found";
      switch(Main.evaluateConfReg(instruccion))
         {
         case 0 : program.addConfiguration(index,configuration);
                  index = Main.evaluateConfigRegWriteCRC(program, instruccion, index);
                  if(index == -1)
                     return;
                  configuration = program.getConfiguration(index);
                  break;
         case 1 : program.addConfiguration(index,configuration);
                  index = Main.evaluateConfigRegWriteFAR(program, instruccion, index);
                  if(index == -1)
                     return;
                  configuration = program.getConfiguration(index);
                  break;
         case 2 : program.addConfiguration(index,configuration);
                  index = Main.evaluateConfigRegWriteFDRI(program, instruccion, index);
                  if(index == -1)
                     return;
                  configuration = program.getConfiguration(index);
                  break;
         case 3 : configuration += " (FORBIDDEN)";
                  index = Main.evaluateConfigRegWriteFDRO(program, instruccion, index);
                  if(index == -1)
                     return;
                  configuration = program.getConfiguration(index);
                  break;
         case 4 : program.addConfiguration(index,configuration);
                  index = Main.evaluateConfigRegWriteCMD(program, instruccion, index);
                  if(index == -1)
                     return;
                  configuration = program.getConfiguration(index);
                  break;
         case 5 : program.addConfiguration(index,configuration);
                  index = Main.evaluateConfigRegWriteCTL(program, instruccion, index);
                  if(index == -1)
                     return;
                  configuration = program.getConfiguration(index);
                  break;
         case 6 : program.addConfiguration(index,configuration);
                  index = Main.evaluateConfigRegWriteMASK(program, instruccion, index);
```

```
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 7 : configuration += " (FORBIDDEN)";
        index = Main.evaluateConfigRegWriteSTAT(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 8 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteLOUT(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 9 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteCOR(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 10 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteMFWR(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 11 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteFLR(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 12 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteKEY(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 13 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteCBC(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
    case 14 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteIDCODE(program, instruccion, index);
        if(index == -1)
            return;
```

```
                configuration = program.getConfiguration(index);
                break;
            default : break;
            }
        }
    else if(Main.evaluateWR(instruccion) == 1)
        {
        int confReg = Main.evaluateConfReg(instruccion);
        if(Main.evaluateConfReg(instruccion) < Main.CONFIGREG.length)


            configuration += "to " + Main.CONFIGREG[Main.evaluateConfReg(instruccion)];
        else
            configuration += "to Not Configuration Register Found";
        String configurationForbidden = "";
        switch(Main.evaluateConfReg(instruccion))
            {
            case 0 : break;
            case 1 : break;
            case 2 : configurationForbidden = " (FORBIDDEN)";
                break;
            case 3 : break;
            case 4 : break;
            case 5 : break;
            case 6 : break;
            case 7 : break;
            case 8 : configurationForbidden = " (FORBIDDEN)";
                break;
            case 9 : break;
            case 10 : configurationForbidden = " (FORBIDDEN)";
                break;
            case 11 : break;
            case 12 : configurationForbidden = " (FORBIDDEN)";
                break;
            case 13 : configurationForbidden = " (FORBIDDEN)";
                break;
            case 14 : break;
            default : break;
            }
        if(Main.evaluateTypeWord(instruccion) == 0)
            {
            configuration += configurationForbidden;
            program.addConfiguration(index,configuration);
            index++;
            if(index < program.getSize())
                instruccion = program.getInstruccion(index);
            else
                return;
            if(Main.evaluatePacket(instruccion) != 2)
                if(Main.evaluateWR(instruccion) != 1)
                    {
```

```
                    index--;
                    break;
                    }
                configuration = "Type " + Main.evaluatePacket(instruccion) +
                    Main.READWRITE[Main.evaluateWR(instruccion)] +
                    Main.evaluateTypeWord(instruccion) + " Words of " +
                    Main.CONFIGREG[confReg] + configurationForbidden;
                program.addConfiguration(index, configuration);
                }
            }
        }
      else
         configuration += "Not Command Found";
      configuration += "\n";
      program.addConfiguration(index,configuration);
      index++;
      }
   }

private static int evaluatePacket(Instruccion instruccion)
   {
   int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINPACKETBITS - 1;
   int endIndex = instruccion.getInstruccionBin().length() - Main.ENDPACKETBITS;
   return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
   }

private static int evaluateTypeWord(Instruccion instruccion)
   {
   int beginIndex;
   int endIndex;
   if(Main.evaluatePacket(instruccion) == 1)
      {
      beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINTYPE1WORD - 1;
      endIndex = instruccion.getInstruccionBin().length() - Main.ENDTYPE1WORD;
      }
   else
      {
      beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINTYPE2WORD - 1;
      endIndex = instruccion.getInstruccionBin().length() - Main.ENDTYPE2WORD;
      }
   return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
   }

private static int evaluateWR(Instruccion instruccion)
   {
   int beginIndex = instruccion.getInstruccionBin().length() - Main.WRITEBIT - 1;
   int endIndex = instruccion.getInstruccionBin().length() - Main.WRITEBIT;
   if(instruccion.getInstruccionBin().substring(beginIndex,endIndex).compareTo("1") == 0)
      return 2;
   beginIndex = instruccion.getInstruccionBin().length() - Main.READBIT - 1;
```

```
    endIndex = instruccion.getInstruccionBin().length() - Main.READBIT;
    if(instruccion.getInstruccionBin().substring(beginIndex, endIndex).compareTo("1") == 0)
      return 1;
    return 0;
    }

  private static int evaluateConfReg(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINCONFIGREG - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.ENDCONFIGREG;
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
    }

  private static int evaluateCMD(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINCMD - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.ENDCMD;
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex, endIndex), 2);
    }

  private static String evaluateID(Instruccion instruccion)
    {
    if(instruccion.getInstruccionHex().compareTo("01226093") == 0)
      return "XC2VP2";
    if(instruccion.getInstruccionHex().compareTo("0123E093") == 0)
      return "XC2VP4";
    if(instruccion.getInstruccionHex().compareTo("0124A093") == 0)
      return "XC2VP7";
    if(instruccion.getInstruccionHex().compareTo("01266093") == 0)
      return "XC2VP20";
    if(instruccion.getInstruccionHex().compareTo("01866093") == 0)
      return "XC2VPX20";
    if(instruccion.getInstruccionHex().compareTo("0127E093") == 0)
      return "XC2VP30";
    if(instruccion.getInstruccionHex().compareTo("01292093") == 0)
      return "XC2VP40";
    if(instruccion.getInstruccionHex().compareTo("0129E093") == 0)
      return "XC2VP50";
    if(instruccion.getInstruccionHex().compareTo("012BA093") == 0)
      return "XC2VP70";
    if(instruccion.getInstruccionHex().compareTo("018BA093") == 0)
      return "XC2VPX70";
    if(instruccion.getInstruccionHex().compareTo("012D6093") == 0)
      return "XC2VP100";
    return "No Virtex-II Found";
    }

  private static int evaluateConfigRegWriteCRC(Program program, Instruccion instruccion, int
index)
    {
```

```
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           if(Main.evaluatePacket(instruccion) != 2)
              if(Main.evaluateWR(instruccion) != 2)
                 return index--;
           configuration = "Type " + Main.evaluatePacket(instruccion) +
                 Main.READWRITE[Main.evaluateWR(instruccion)] +
                 Main.evaluateTypeWord(instruccion) + " Words to CRC";
           program.addConfiguration(index, configuration);
           }
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           configuration = "Packet Data: Word " + i + " CRC Register = 0x" +
                 instruccion.getInstruccionHex();
           program.addConfiguration(index, configuration);
           }
        return index;
        }

     private static int evaluateConfigRegWriteFAR(Program program, Instruccion instruccion, int
index)
        {
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           if(Main.evaluatePacket(instruccion) != 2)
              if(Main.evaluateWR(instruccion) != 2)
                 return index--;
           configuration = "Type " + Main.evaluatePacket(instruccion) +
                 Main.READWRITE[Main.evaluateWR(instruccion)] +
                 Main.evaluateTypeWord(instruccion) + " Words to FAR";
           program.addConfiguration(index, configuration);
           }
```

```
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           configuration = "Packet Data: Word " + i + " Frame Address Register = 0x" +
                 instruccion.getInstruccionHex();
           program.addConfiguration(index, configuration);
           }
        return index;
        }

    private static int evaluateConfigRegWriteFDRI(Program program, Instruccion instruccion, int
index)
        {
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           if(Main.evaluatePacket(instruccion) != 2)
              if(Main.evaluateWR(instruccion) != 2)
                 return index--;
           configuration = "Type " + Main.evaluatePacket(instruccion) +
                 Main.READWRITE[Main.evaluateWR(instruccion)] +
                 Main.evaluateTypeWord(instruccion) + " Words to FDRI";
           program.addConfiguration(index,
 configuration);
           }
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           configuration = "Packet Data: Word " + i + " Frame Data Input Register = 0x" +
                 instruccion.getInstruccionHex();
           program.addConfiguration(index, configuration);
           }
        index++;
        if(index < program.getSize())
           instruccion = program.getInstruccion(index);
```

```
        else
            return -1;
        configuration = "Packet Data: AutoCRC Word = 0x" + instruccion.getInstruccionHex();
        program.addConfiguration(index, configuration);
        return index;
        }

    private static int evaluateConfigRegWriteFDRO(Program program, Instruccion instruccion, int
index)
        {
        String configuration;

        return index;
        }

    private static int evaluateConfigRegWriteCMD(Program program, Instruccion instruccion, int
index)
        {
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
            {
            index++;
            if(index < program.getSize())
                instruccion = program.getInstruccion(index);
            else
                return -1;
            if(Main.evaluatePacket(instruccion) != 2)
                if(Main.evaluateWR(instruccion) != 2)
                    return index--;
            configuration = "Type " + Main.evaluatePacket(instruccion) +
                    Main.READWRITE[Main.evaluateWR(instruccion)] +
                    Main.evaluateTypeWord(instruccion) + " Words to CMD";
            program.addConfiguration(index, configuration);
            }
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
            {
            index++;
            if(index < program.getSize())
                instruccion = program.getInstruccion(index);
            else
                return -1;
            if(Main.evaluateCMD(instruccion) < Main.CMD.length)
                configuration = "Packet Data: Word " + i + " CMD Register = " +
                        Main.CMD[Main.evaluateCMD(instruccion)] + " Command";
            else
                return index--;
            program.addConfiguration(index, configuration);
            }
        return index;
```

```
    }

  private static int evaluateConfigRegWriteCTL(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      if(Main.evaluatePacket(instruccion) != 2)
        if(Main.evaluateWR(instruccion) != 2)
          return index--;
      configuration = "Type " + Main.evaluatePacket(instruccion) +
          Main.READWRITE[Main.evaluateWR(instruccion)] +
          Main.evaluateTypeWord(instruccion) + " Words to CTL";
      program.addConfiguration(index, configuration);
      }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      configuration = "Packet Data: Word " + i + " Control Register = 0x" +
          instruccion.getInstruccionHex();
      program.addConfiguration(index, configuration);
      }
    return index;
    }

  private static int evaluateConfigRegWriteMASK(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      if(Main.evaluatePacket(instruccion) != 2)
        if(Main.evaluateWR(instruccion) != 2)
          return index--;
```

```
            configuration = "Type " + Main.evaluatePacket(instruccion) +
                  Main.READWRITE[Main.evaluateWR(instruccion)] +
                  Main.evaluateTypeWord(instruccion) + " Words to MASK";
            program.addConfiguration(index, configuration);
            }
         int numWords = Main.evaluateTypeWord(instruccion);
         for(int i = 1 ; i <= numWords ; i++)
            {
            index++;
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return -1;
            configuration = "Packet Data: Word " + i + " Mask Register = 0x" +
                  instruccion.getInstruccionHex();
            program.addConfiguration(index, configuration);
            }
         return index;
         }

      private static int evaluateConfigRegWriteSTAT(Program program, Instruccion instruccion, int
index)
         {
         String configuration;

         return index;
         }

      private static int evaluateConfigRegWriteLOUT(Program program, Instruccion instruccion, int
index)
         {
         String configuration;
         if(Main.evaluateTypeWord(instruccion) == 0)
            {
            index++;
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return -1;
            if(Main.evaluatePacket(instruccion) != 2)
               if(Main.evaluateWR(instruccion) != 2)
                  return index--;
            configuration = "Type " + Main.evaluatePacket(instruccion) +
                  Main.READWRITE[Main.evaluateWR(instruccion)] +
                  Main.evaluateTypeWord(instruccion) + " Words to LOUT";
            program.addConfiguration(index, configuration);
            }
         int numWords = Main.evaluateTypeWord(instruccion);
         for(int i = 1 ; i <= numWords ; i++)
            {
```

```
            index++;
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return -1;
            configuration = "Packet Data: Word " + i + " Legacy Output Register = 0x" +
                  instruccion.getInstruccionHex();
            program.addConfiguration(index, configuration);
            }
         return index;
         }

      private static int evaluateConfigRegWriteCOR(Program program, Instruccion instruccion, int
index)
         {
         String configuration;
         if(Main.evaluateTypeWord(instruccion) == 0)
            {
            index++;
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return -1;
            if(Main.evaluatePacket(instruccion) != 2)
               if(Main.evaluateWR(instruccion) != 2)
                  return index--;
            configuration = "Type " + Main.evaluatePacket(instruccion) +
                  Main.READWRITE[Main.evaluateWR(instruccion)] +
                  Main.evaluateTypeWord(instruccion) + " Words to COR";
            program.addConfiguration(index, configuration);
            }
         int numWords = Main.evaluateTypeWord(instruccion);
         for(int i = 1 ; i <= numWords ; i++)
            {
            index++;
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return -1;
            configuration = "Packet Data: Word " + i + " Configuration Options Register = 0x" +
                  instruccion.getInstruccionHex();
            program.addConfiguration(index, configuration);
            }
         return index;
         }

      private static int evaluateConfigRegWriteMFWR(Program program, Instruccion instruccion, int
index)
         {
         String configuration;
```

```
    if(Main.evaluateTypeWord(instruccion) == 0)
       {
       index++;
       if(index < program.getSize())
          instruccion = program.getInstruccion(index);
       else
          return -1;
       if(Main.evaluatePacket(instruccion) != 2)
          if(Main.evaluateWR(instruccion) != 2)
             return index--;
       configuration = "Type " + Main.evaluatePacket(instruccion) +
             Main.READWRITE[Main.evaluateWR(instruccion)] +
             Main.evaluateTypeWord(instruccion) + " Words to MFWR";
       program.addConfiguration(index, configuration);
       }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
       {
       index++;
       if(index < program.getSize())
          instruccion = program.getInstruccion(index);
       else
          return -1;
       configuration = "Packet Data: Word " + i + " Multiple Frame Write Register = 0x" +
             instruccion.getInstruccionHex();
       program.addConfiguration(index, configuration);
       }
    return index;
    }

  private static int evaluateConfigRegWriteFLR(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
       {
       index++;
       if(index < program.getSize())
          instruccion = program.getInstruccion(index);
       else
          return -1;
       if(Main.evaluatePacket(instruccion) != 2)
          if(Main.evaluateWR(instruccion) != 2)


      return index--;
       configuration = "Type " + Main.evaluatePacket(instruccion) +
             Main.READWRITE[Main.evaluateWR(instruccion)] +
             Main.evaluateTypeWord(instruccion) + " Words to FLR";
       program.addConfiguration(index, configuration);
       }
```

```
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           configuration = "Packet Data: Word " + i + " Frame Length Register = " +
                 Integer.parseInt(instruccion.getInstruccionBin(),2) +
                 " Words = 0x" + instruccion.getInstruccionHex();
           program.addConfiguration(index, configuration);
           }
        return index;
        }

    private static int evaluateConfigRegWriteKEY(Program program, Instruccion instruccion, int
index)
        {
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           if(Main.evaluatePacket(instruccion) != 2)
              if(Main.evaluateWR(instruccion) != 2)
                 return index--;
           configuration = "Type " + Main.evaluatePacket(instruccion) +
                 Main.READWRITE[Main.evaluateWR(instruccion)] +
                 Main.evaluateTypeWord(instruccion) + " Words to KEY";
           program.addConfiguration(index, configuration);
           }
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           configuration = "Packet Data: Word " + i + " Initial Key Address Register = 0x" +
                 instruccion.getInstruccionHex();
           program.addConfiguration(index, configuration);
           }
        return index;
        }
```

```
   private static int evaluateConfigRegWriteCBC(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
               return index--;
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words to CBC";
         program.addConfiguration(index, configuration);
         }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         configuration = "Packet Data: Word " + i + " Cipher Block Chaining Register = 0x" +
               instruccion.getInstruccionHex();
         program.addConfiguration(index, configuration);
         }
      return index;
      }

   private static int evaluateConfigRegWriteIDCODE(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
               return index--;
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
```

```
                Main.evaluateTypeWord(instruccion) + " Words to IDCODE";
            program.addConfiguration(index, configuration);
            }
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
            {
            index++;
            if(index < program.getSize())
                instruccion = program.getInstruccion(index);
            else
                return -1;
            configuration = "Packet Data: Word " + i + " Device ID = 0x" +
                    instruccion.getInstruccionHex() + " (" + Main.evaluateID(instruccion) + ")";
            program.addConfiguration(index, configuration);
            }
        return index;
        }

}
```

Read.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package IO;

import Program.*;
import java.io.*;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Read
{

    private String inputFileName;

    public Read (String inputFileName)
        {
```

```
        this.inputFileName = inputFileName;
    }

  public void readFile(Program program)
    {
    DataInputStream dis = null;
    String hexRubish;
    byte[] hex = new byte[4];
    try
      {
      FileInputStream fis = new FileInputStream(this.inputFileName);
      boolean rubish = true;
      dis = new DataInputStream(fis);
      hexRubish = Integer.toHexString(dis.readUnsignedByte());
      while(rubish)
        {
        while(0 != hexRubish.compareTo("aa"))
          hexRubish = Integer.toHexString(dis.readUnsignedByte());
        if(0 == ((hexRubish = Integer.toHexString(dis.readUnsignedByte())).compareTo("99")))
          if(0 == ((hexRubish =
Integer.toHexString(dis.readUnsignedByte())).compareTo("55")))
            if(0 == ((hexRubish =
Integer.toHexString(dis.readUnsignedByte())).compareTo("66")))
              rubish = false;
        }
      hex = new byte[]{(byte)0xAA, (byte)0x99, (byte)0x55, (byte)0x66};
      Instruccion instruccion = new Instruccion(hex);
      program.addInstruccion(instruccion);
      while(true)
        {
        if(dis.read(hex,0,4) == -1)
          throw new EOFException();
        instruccion = new Instruccion(hex);
        program.addInstruccion(instruccion);
        }
      }
    catch(EOFException eofe)
      {
      try
        {
        dis.close();
        }
      catch(IOException ioe)
        {
        System.err.println("IOException: " + ioe.getMessage());
        }
      }
    catch(IOException ioe)
      {
      System.err.println("IOException: " + ioe.getMessage());
```

```
        }
    }

}
```

Write.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package IO;

import Program.Program;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Write
{

   private String outputFileName;
   private FileWriter fileWriter;
   private PrintWriter bufferOut;

   public Write (String outputFileName)
      {
      this.outputFileName = outputFileName;
      }

   public void writeProgram(Program program)
      {
      try
         {
         this.fileWriter = new FileWriter(this.outputFileName + ".v2cv");
         this.bufferOut = new PrintWriter(this.fileWriter);
         for(int index = 0 ; index < program.getSize() ; index++)
```

```
            {
            //System.out.print(program.getInstruccion(index).getInstruccionHex());
            //System.out.println(" - " + program.getInstruccion(index).getInstruccionBin());
            //System.out.println(program.getConfiguration(index));
            this.bufferOut.print(program.getInstruccion(index).getInstruccionHex());
            this.bufferOut.println(" - " + program.getInstruccion(index).getInstruccionBin());
            this.bufferOut.println(program.getConfiguration(index));
            }
         this.bufferOut.close();
         this.fileWriter.close();
         }
      catch (IOException ioe)
         {
         System.err.println("IOException: " + ioe.getMessage());
         }
      }

}
```

Instruccion.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Program;

import java.io.ByteArrayInputStream;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Instruccion
{

   private byte[] instruccion;
   private String instruccionBin;
   private String instruccionHex;

   public Instruccion(byte[] instruccion)
```

```
    {
    this.instruccion = new byte[4];
    this.instruccion[0] = instruccion[0];
    this.instruccion[1] = instruccion[1];
    this.instruccion[2] = instruccion[2];
    this.instruccion[3] = instruccion[3];
    this.instruccionBin = this.toBinary(this.instruccion);
    this.instruccionHex = this.toHexadecimal(this.instruccion).toUpperCase();
    }

public byte[] getInstruccion()
    {
    return this.instruccion;
    }

public String getInstruccionBin()
    {
    return this.instruccionBin;
    }

public String getInstruccionHex()
    {
    return this.instruccionHex;
    }

private String toHexadecimal(byte[] hex)
    {
        String result="";
        ByteArrayInputStream input = new ByteArrayInputStream(hex);
        String resultAux;
        int read = input.read();
        while(read != -1)
        {
            resultAux = Integer.toHexString(read);
            if(resultAux.length() < 2)
                result += "0";
            result += resultAux;
            read = input.read();
        }
        return result.toUpperCase();
    }

private String toBinary(byte[] hex)
    {
        String result="";
        ByteArrayInputStream input = new ByteArrayInputStream(hex);
        String resultAux;
        int read = input.read();
        while(read != -1)
        {
```

```
                resultAux = Integer.toBinaryString(read);
                while(resultAux.length() < 8)
                   resultAux = "0" + resultAux;
                result += resultAux;
                read = input.read();
            }
            return result;
        }

}
```

Program.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Program;

import java.util.ArrayList;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Program
{

   private ArrayList program;
   private ArrayList configuration;

   public Program()
      {
      this.program = new ArrayList();
      this.configuration = new ArrayList();
      }

   public void addInstruccion(Instruccion instruccion)
      {
      this.program.add(instruccion);
      }
```

```
    public void addConfiguration(String configuration)
        {
        this.configuration.add(configuration);
        }

    public void addConfiguration(int index, String configuration)
        {
        this.configuration.add(index, configuration);
        }

    public Instruccion getInstruccion(int index)
        {
        return (Instruccion)this.program.get(index);
        }

    public String getConfiguration(int index)
        {
        return (String)this.configuration.get(index);
        }

    public int getSize()
        {
        return this.program.size();
        }

}
```

Código Virtex II Configuration Comparer

Main.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package virtexiiconfigurationcomparer;

import IO.*;
import Program.*;
import Comparision.*;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Main
{

    /**
     * @param args the command line arguments
     */

    private final static int BEGINPACKETBITS = 31;
    private final static int ENDPACKETBITS = 29;
    private final static int WRITEBIT = 28;
    private final static int READBIT = 27;
    private final static int BEGINTYPE1WORD = 10;
    private final static int ENDTYPE1WORD = 0;
    private final static int BEGINTYPE2WORD = 26;
    private final static int ENDTYPE2WORD = 0;
    private final static int BEGINCONFIGREG = 17;
    private final static int ENDCONFIGREG = 13;
    private final static int BEGINCMD = 3;
    private final static int ENDCMD = 0;
    private final static String[] READWRITE = {" "," Read ", " Write "};
    private final static String[] CONFIGREG = {"CRC","FAR","FDRI","FDRO","CMD",
        "CTL","MASK","STAT","LOUT","COR","MFWR","FLR","KEY","CBC","IDCODE"};
    private final static String[] CMD = {" ","WCFG","MFWR","LFRM","RCFG","START",
```

```
    "RCAP","RCRC","AGHIGH","SWITCH","GRESTORE","SHUTDOWN","GCAPTURE","D
ESYNCH"};

   private static Read input1;
   private static Read input2;
   private static Write output1;
   private static Write output2;
   private static Write output3;

   public static void main(String[] args)
      {
      // TODO code application logic here

      Main.input1 = new Read(args[0]);
      Main.input2 = new Read(args[1]);
      Main.output1 = new Write(args[0]);
      Main.output2 = new Write(args[1]);
      Main.output3 = new Write(args[0] + "_" + args[1] + "_Comparision");
      Program program1 = new Program(args[0]);
      Program program2 = new Program(args[1]);
      Main.input1.readFile(program1);
      Main.input2.readFile(program2);
      Main.translate(program1);
      Main.translate(program2);
      Main.output1.writeProgram(program1);
      Main.output2.writeProgram(program2);
      Comparision comparision = new Comparision();
      Main.compare(program1, program2, comparision);
      Main.output3.writeComparision(program1, program2, comparision);
      }

   private static void translate(Program program)
      {
      Instruccion instruccion;
      int index = 0;
      String configuration;
      while(index < program.getSize())
         {
         instruccion = program.getInstruccion(index);
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words ";
         if(instruccion.getInstruccionHex().compareTo("AA995566") == 0)
            {
            configuration = "Sync Word";
            }
         else if(instruccion.getInstruccionHex().compareTo("20000000") == 0)
            {
            int count = 1;
            while(instruccion.getInstruccionHex().compareTo("20000000") == 0)
```

```
        {
        configuration = "Type " + Main.evaluatePacket(instruccion) +
              " NOOP Word " + count;
        program.addConfiguration(index,configuration);
        count++;
        index++;
        if(index < program.getSize())
           instruccion = program.getInstruccion(index);
        else
           return;
        }
     index--;
     }
  else if(Main.evaluatePacket(instruccion) == 1)
     {
     if(Main.evaluateWR(instruccion) == 2)
        {
        if(Main.evaluateConfReg(instruccion) < Main.CONFIGREG.length)
           configuration += "to " + Main.CONFIGREG[Main.evaluateConfReg(instruccion)];
        else
           configuration += "to Not Configuration Register Found";
        switch(Main.evaluateConfReg(instruccion))
           {
           case 0 : program.addConfiguration(index,configuration);
                 index = Main.evaluateConfigRegWriteCRC(program, instruccion, index);
                 if(index == -1)
                    return;
                 configuration = program.getConfiguration(index);
                 break;
           case 1 : program.addConfiguration(index,configuration);
                 index = Main.evaluateConfigRegWriteFAR(program, instruccion, index);
                 if(index == -1)
                    return;
                 configuration = program.getConfiguration(index);
                 break;
           case 2 : program.addConfiguration(index,configuration);
                 index = Main.evaluateConfigRegWriteFDRI(program, instruccion, index);
                 if(index == -1)
                    return;
                 configuration = program.getConfiguration(index);
                 break;
           case 3 : configuration += " (FORBIDDEN)";
                 index = Main.evaluateConfigRegWriteFDRO(program, instruccion, index);
                 if(index == -1)
                    return;
                 configuration = program.getConfiguration(index);
                 break;
           case 4 : program.addConfiguration(index,configuration);
                 index = Main.evaluateConfigRegWriteCMD(program, instruccion, index);
                 if(index == -1)
```

```
            return;
        configuration = program.getConfiguration(index);
        break;
case 5 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteCTL(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
case 6 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteMASK(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
case 7 : configuration += " (FORBIDDEN)";
        index = Main.evaluateConfigRegWriteSTAT(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
case 8 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteLOUT(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
case 9 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteCOR(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
case 10 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteMFWR(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
case 11 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteFLR(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
        break;
case 12 : program.addConfiguration(index,configuration);
        index = Main.evaluateConfigRegWriteKEY(program, instruccion, index);
        if(index == -1)
            return;
        configuration = program.getConfiguration(index);
```

```
            break;
        case 13 : program.addConfiguration(index,configuration);
            index = Main.evaluateConfigRegWriteCBC(program, instruccion, index);
            if(index == -1)
               return;
            configuration = program.getConfiguration(index);
            break;
        case 14 : program.addConfiguration(index,configuration);


            index = Main.evaluateConfigRegWriteIDCODE(program, instruccion, index);
            if(index == -1)
               return;
            configuration = program.getConfiguration(index);
            break;
        default : break;
        }
    }
else if(Main.evaluateWR(instruccion) == 1)
    {
    int confReg = Main.evaluateConfReg(instruccion);
    if(Main.evaluateConfReg(instruccion) < Main.CONFIGREG.length)
        configuration += "to " + Main.CONFIGREG[Main.evaluateConfReg(instruccion)];
    else
        configuration += "to Not Configuration Register Found";
    String configurationForbidden = "";
    switch(Main.evaluateConfReg(instruccion))
        {
        case 0 : break;
        case 1 : break;
        case 2 : configurationForbidden = " (FORBIDDEN)";
            break;
        case 3 : break;
        case 4 : break;
        case 5 : break;
        case 6 : break;
        case 7 : break;
        case 8 : configurationForbidden = " (FORBIDDEN)";
            break;
        case 9 : break;
        case 10 : configurationForbidden = " (FORBIDDEN)";
            break;
        case 11 : break;
        case 12 : configurationForbidden = " (FORBIDDEN)";
            break;
        case 13 : configurationForbidden = " (FORBIDDEN)";
            break;
        case 14 : break;
        default : break;
        }
    if(Main.evaluateTypeWord(instruccion) == 0)
```

```
                {
                configuration += configurationForbidden;
                program.addConfiguration(index,configuration);
                index++;
                if(index < program.getSize())
                   instruccion = program.getInstruccion(index);
                else
                   return;
                if(Main.evaluatePacket(instruccion) != 2)
                   if(Main.evaluateWR(instruccion) != 1)
                      {
                      index--;
                      break;
                      }
                configuration = "Type " + Main.evaluatePacket(instruccion) +
                      Main.READWRITE[Main.evaluateWR(instruccion)] +
                      Main.evaluateTypeWord(instruccion) + " Words of " +
                      Main.CONFIGREG[confReg] + configurationForbidden;
                program.addConfiguration(index, configuration);
                }
             }
          }
       else
          configuration += "Not Command Found";
       configuration += "\n";
       program.addConfiguration(index,configuration);
       index++;
       }
    }

private static int evaluatePacket(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINPACKETBITS - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.ENDPACKETBITS;
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
    }

private static int evaluateTypeWord(Instruccion instruccion)
    {
    int beginIndex;
    int endIndex;
    if(Main.evaluatePacket(instruccion) == 1)
       {
       beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINTYPE1WORD - 1;
       endIndex = instruccion.getInstruccionBin().length() - Main.ENDTYPE1WORD;
       }
    else
       {
       beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINTYPE2WORD - 1;
       endIndex = instruccion.getInstruccionBin().length() - Main.ENDTYPE2WORD;
```

```
        }
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
    }

private static int evaluateWR(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.WRITEBIT - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.WRITEBIT;
    if(instruccion.getInstruccionBin().substring(beginIndex,endIndex).compareTo("1") == 0)
        return 2;
    beginIndex = instruccion.getInstruccionBin().length() - Main.READBIT - 1;
    endIndex = instruccion.getInstruccionBin().length() - Main.READBIT;
    if(instruccion.getInstruccionBin().substring(beginIndex, endIndex).compareTo("1") == 0)
        return 1;
    return 0;
    }

private static int evaluateConfReg(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINCONFIGREG - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.ENDCONFIGREG;
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
    }

private static int evaluateCMD(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINCMD - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.ENDCMD;
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex, endIndex), 2);
    }

private static String evaluateID(Instruccion instruccion)
    {
    if(instruccion.getInstruccionHex().compareTo("01226093") == 0)
        return "XC2VP2";
    if(instruccion.getInstruccionHex().compareTo("0123E093") == 0)
        return "XC2VP4";
    if(instruccion.getInstruccionHex().compareTo("0124A093") == 0)
        return "XC2VP7";
    if(instruccion.getInstruccionHex().compareTo("01266093") == 0)
        return "XC2VP20";
    if(instruccion.getInstruccionHex().compareTo("01866093") == 0)
        return "XC2VPX20";
    if(instruccion.getInstruccionHex().compareTo("0127E093") == 0)
        return "XC2VP30";
    if(instruccion.getInstruccionHex().compareTo("01292093") == 0)
        return "XC2VP40";
    if(instruccion.getInstruccionHex().compareTo("0129E093") == 0)
        return "XC2VP50";
    if(instruccion.getInstruccionHex().compareTo("012BA093") == 0)
```

```
        return "XC2VP70";
    if(instruccion.getInstruccionHex().compareTo("018BA093") == 0)
        return "XC2VPX70";
    if(instruccion.getInstruccionHex().compareTo("012D6093") == 0)
        return "XC2VP100";
    return "No Virtex-II Found";
    }

  private static int evaluateConfigRegWriteCRC(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
                return index--;
        configuration = "Type " + Main.evaluatePacket(instruccion) +
                Main.READWRITE[Main.evaluateWR(instruccion)] +
                Main.evaluateTypeWord(instruccion) + " Words to CRC";
        program.addConfiguration(index, configuration);
        }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        configuration = "Packet Data: Word " + i + " CRC Register = 0x" +
                instruccion.getInstruccionHex();
        program.addConfiguration(index, configuration);
        }
    return index;
    }

  private static int evaluateConfigRegWriteFAR(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
```

```
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
               return index--;
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words to FAR";
         program.addConfiguration(index, configuration);
         }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         configuration = "Packet Data: Word " + i + " Frame Address Register = 0x" +
               instruccion.getInstruccionHex();
         program.addConfiguration(index, configuration);
         }
      return index;
      }

   private static int evaluateConfigRegWriteFDRI(Program program, Instruccion instruccion, int
index)
      {
      String
 configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
               return index--;
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words to FDRI";
         program.addConfiguration(index, configuration);
         }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
         {
         index++;
```

```
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return -1;
            configuration = "Packet Data: Word " + i + " Frame Data Input Register = 0x" +
                  instruccion.getInstruccionHex();
            program.addConfiguration(index, configuration);
            }
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         configuration = "Packet Data: AutoCRC Word = 0x" + instruccion.getInstruccionHex();
         program.addConfiguration(index, configuration);
         return index;
         }

      private static int evaluateConfigRegWriteFDRO(Program program, Instruccion instruccion, int
index)
         {
         String configuration;

         return index;
         }

      private static int evaluateConfigRegWriteCMD(Program program, Instruccion instruccion, int
index)
         {
         String configuration;
         if(Main.evaluateTypeWord(instruccion) == 0)
            {
            index++;
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return -1;
            if(Main.evaluatePacket(instruccion) != 2)
               if(Main.evaluateWR(instruccion) != 2)
                  return index--;
            configuration = "Type " + Main.evaluatePacket(instruccion) +
                  Main.READWRITE[Main.evaluateWR(instruccion)] +
                  Main.evaluateTypeWord(instruccion) + " Words to CMD";
            program.addConfiguration(index, configuration);
            }
         int numWords = Main.evaluateTypeWord(instruccion);
         for(int i = 1 ; i <= numWords ; i++)
            {
            index++;
            if(index < program.getSize())
```

```
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluateCMD(instruccion) < Main.CMD.length)
            configuration = "Packet Data: Word " + i + " CMD Register = " +
                  Main.CMD[Main.evaluateCMD(instruccion)] + " Command";
         else
            return index--;
         program.addConfiguration(index, configuration);
         }
      return index;
      }

   private static int evaluateConfigRegWriteCTL(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
               return index--;
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words to CTL";
         program.addConfiguration(index, configuration);
         }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         configuration = "Packet Data: Word " + i + " Control Register = 0x" +
               instruccion.getInstruccionHex();
         program.addConfiguration(index, configuration);
         }
      return index;
      }

   private static int evaluateConfigRegWriteMASK(Program program, Instruccion instruccion, int
index)
      {
```

```
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           if(Main.evaluatePacket(instruccion) != 2)
              if(Main.evaluateWR(instruccion) != 2)
                 return index--;
           configuration = "Type " + Main.evaluatePacket(instruccion) +
                 Main.READWRITE[Main.evaluateWR(instruccion)] +
                 Main.evaluateTypeWord(instruccion) + " Words to MASK";
           program.addConfiguration(index, configuration);
           }
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
           configuration = "Packet Data: Word " + i + " Mask Register = 0x" +
                 instruccion.getInstruccionHex();
           program.addConfiguration(index, configuration);
           }
        return index;
        }

     private static int evaluateConfigRegWriteSTAT(Program program, Instruccion instruccion, int
index)
        {
        String configuration;

        return index;
        }

     private static int evaluateConfigRegWriteLOUT(Program program, Instruccion instruccion, int
index)
        {
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
           {
           index++;
           if(index < program.getSize())
              instruccion = program.getInstruccion(index);
           else
              return -1;
```

```
        if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
                return index--;
        configuration = "Type " + Main.evaluatePacket(instruccion) +
                Main.READWRITE[Main.evaluateWR(instruccion)] +
                Main.evaluateTypeWord(instruccion) + " Words to LOUT";
        program.addConfiguration(index, configuration);
        }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        configuration = "Packet Data: Word " + i + " Legacy Output Register = 0x" +
                instruccion.getInstruccionHex();
        program.addConfiguration(index, configuration);
        }
      return index;
      }

   private static int evaluateConfigRegWriteCOR(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
                return index--;
        configuration = "Type " + Main.evaluatePacket(instruccion) +
                Main.READWRITE[Main.evaluateWR(instruccion)] +
                Main.evaluateTypeWord(instruccion) + " Words to COR";
        program.addConfiguration(index, configuration);
        }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
```

```
        configuration = "Packet Data: Word " + i + " Configuration Options Register = 0x" +
              instruccion.getInstruccionHex();
        program.addConfiguration(index, configuration);
        }
      return index;
      }

   private static int evaluateConfigRegWriteMFWR(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
          instruccion = program.getInstruccion(index);
        else
          return -1;
        if(Main.evaluatePacket(instruccion) != 2)
          if(Main.evaluateWR(instruccion) != 2)
            return index--;
        configuration = "Type " + Main.evaluatePacket(instruccion) +
              Main.READWRITE[Main.evaluateWR(instruccion)] +
              Main.evaluateTypeWord(instruccion) + " Words to MFWR";
        program.addConfiguration(index, configuration);
        }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
          instruccion = program.getInstruccion(index);
        else
          return -1;
        configuration = "Packet Data: Word " + i + " Multiple Frame Write Register = 0x" +


            instruccion.getInstruccionHex();
        program.addConfiguration(index, configuration);
        }
      return index;
      }

   private static int evaluateConfigRegWriteFLR(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
```

```
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
               return index--;
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words to FLR";
         program.addConfiguration(index, configuration);
         }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         configuration = "Packet Data: Word " + i + " Frame Length Register = " +
               Integer.parseInt(instruccion.getInstruccionBin(),2) +
               " Words = 0x" + instruccion.getInstruccionHex();
         program.addConfiguration(index, configuration);
         }
      return index;
      }

   private static int evaluateConfigRegWriteKEY(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
         {
         index++;
         if(index < program.getSize())
            instruccion = program.getInstruccion(index);
         else
            return -1;
         if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
               return index--;
         configuration = "Type " + Main.evaluatePacket(instruccion) +
               Main.READWRITE[Main.evaluateWR(instruccion)] +
               Main.evaluateTypeWord(instruccion) + " Words to KEY";
         program.addConfiguration(index, configuration);
         }
      int numWords = Main.evaluateTypeWord(instruccion);
      for(int i = 1 ; i <= numWords ; i++)
         {
         index++;
```

```
        if(index < program.getSize())
           instruccion = program.getInstruccion(index);
        else
           return -1;
        configuration = "Packet Data: Word " + i + " Initial Key Address Register = 0x" +
              instruccion.getInstruccionHex();
        program.addConfiguration(index, configuration);
        }
     return index;
     }

   private static int evaluateConfigRegWriteCBC(Program program, Instruccion instruccion, int
index)
     {
     String configuration;
     if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
           instruccion = program.getInstruccion(index);
        else
           return -1;
        if(Main.evaluatePacket(instruccion) != 2)
           if(Main.evaluateWR(instruccion) != 2)
              return index--;
        configuration = "Type " + Main.evaluatePacket(instruccion) +
              Main.READWRITE[Main.evaluateWR(instruccion)] +
              Main.evaluateTypeWord(instruccion) + " Words to CBC";
        program.addConfiguration(index, configuration);
        }
     int numWords = Main.evaluateTypeWord(instruccion);
     for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
           instruccion = program.getInstruccion(index);
        else
           return -1;
        configuration = "Packet Data: Word " + i + " Cipher Block Chaining Register = 0x" +
              instruccion.getInstruccionHex();
        program.addConfiguration(index, configuration);
        }
     return index;
     }

   private static int evaluateConfigRegWriteIDCODE(Program program, Instruccion instruccion, int
index)
     {
     String configuration;
     if(Main.evaluateTypeWord(instruccion) == 0)
```

```
        {
        index++;
        if(index < program.getSize())
           instruccion = program.getInstruccion(index);
        else
           return -1;
        if(Main.evaluatePacket(instruccion) != 2)
           if(Main.evaluateWR(instruccion) != 2)
              return index--;
        configuration = "Type " + Main.evaluatePacket(instruccion) +
              Main.READWRITE[Main.evaluateWR(instruccion)] +
              Main.evaluateTypeWord(instruccion) + " Words to IDCODE";
        program.addConfiguration(index, configuration);
        }
     int numWords = Main.evaluateTypeWord(instruccion);
     for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
           instruccion = program.getInstruccion(index);
        else
           return -1;
        configuration = "Packet Data: Word " + i + " Device ID = 0x" +
              instruccion.getInstruccionHex() + " (" + Main.evaluateID(instruccion) + ")";
        program.addConfiguration(index, configuration);
        }
     return index;
     }

  private static void compare(Program program1, Program program2, Comparision comparision)
     {
     Instruccion instruccion1;
     Instruccion instruccion2;
     int index = 0;
     while((index < program1.getSize()) & (index < program2.getSize()))
        {
        instruccion1 = program1.getInstruccion(index);
        instruccion2 = program2.getInstruccion(index);
        if(instruccion1.getInstruccionHex().compareTo(instruccion2.getInstruccionHex()) != 0)
           comparision.addComparision(index);
        index++;
        }
     }

}
```

Comparision.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Comparision;

import java.util.ArrayList;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Comparision
{

   private ArrayList comparision;

   public Comparision()
      {
      this.comparision = new ArrayList();
      }

   public void addComparision(int comparision)
      {
      this.comparision.add(comparision);
      }

   public int getComparision(int index)
      {
      return (Integer)this.comparision.get(index);
      }

   public int getSize()
      {
      return this.comparision.size();
      }

}
```

Read.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package IO;

import Program.*;
import java.io.*;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Read
{

   private String inputFileName;

   public Read (String inputFileName)
      {
         this.inputFileName = inputFileName;
      }

   public void readFile(Program program)
      {
      DataInputStream dis = null;
      String hexRubish;
      byte[] hex = new byte[4];
      try
         {
         FileInputStream fis = new FileInputStream(this.inputFileName);
         boolean rubish = true;
         dis = new DataInputStream(fis);
         hexRubish = Integer.toHexString(dis.readUnsignedByte());
         while(rubish)
            {
            while(0 != hexRubish.compareTo("aa"))
               hexRubish = Integer.toHexString(dis.readUnsignedByte());
            if(0 == ((hexRubish = Integer.toHexString(dis.readUnsignedByte())).compareTo("99")))
               if(0 == ((hexRubish =
Integer.toHexString(dis.readUnsignedByte())).compareTo("55")))
```

```
                    if(0 == ((hexRubish =
Integer.toHexString(dis.readUnsignedByte())).compareTo("66")))
                        rubish = false;
                }
            hex = new byte[]{(byte)0xAA, (byte)0x99, (byte)0x55, (byte)0x66};
            Instruccion instruccion = new Instruccion(hex);
            program.addInstruccion(instruccion);
            while(true)
                {
                if(dis.read(hex,0,4) == -1)
                    throw new EOFException();
                instruccion = new Instruccion(hex);
                program.addInstruccion(instruccion);
                }
            }
        catch(EOFException eofe)
            {
            try
                {
                dis.close();
                }
            catch(IOException ioe)
                {
                System.err.println("IOException: " + ioe.getMessage());
                }
            }
        catch(IOException ioe)
            {
            System.err.println("IOException: " + ioe.getMessage());
            }
        }

}
```

Write.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package IO;

import Comparision.Comparision;
import Program.Program;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
```

```
/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Write
{

   private String outputFileName;
   private FileWriter fileWriter;
   private PrintWriter bufferOut;

   public Write (String outputFileName)
      {
      this.outputFileName = outputFileName;
      }

   public void writeProgram(Program program)
      {
      try
         {
         this.fileWriter = new FileWriter(this.outputFileName + ".v2cv");
         this.bufferOut = new PrintWriter(this.fileWriter);
         for(int index = 0 ; index < program.getSize() ; index++)
            {
            //System.out.print(index + " : " + program.getInstruccion(index).getInstruccionHex());
            //System.out.println(" - " + program.getInstruccion(index).getInstruccionBin());
            //System.out.println(program.getConfiguration(index));
            this.bufferOut.print(index + " : " + program.getInstruccion(index).getInstruccionHex());
            this.bufferOut.println(" - " + program.getInstruccion(index).getInstruccionBin());
            this.bufferOut.println(program.getConfiguration(index));
            }
         this.bufferOut.close();
         this.fileWriter.close();
         }
      catch (IOException ioe)
         {
         System.err.println("IOException: " + ioe.getMessage());
         }
      }

   public void writeComparision(Program program1, Program program2, Comparision
comparision)
      {
```

```
        try
            {
            this.fileWriter = new FileWriter(this.outputFileName + ".v2cc");
            this.bufferOut = new PrintWriter(this.fileWriter);
            for(int index = 0 ; index < comparision.getSize() ; index++)
                {
                //System.out.print("Diference at Line : ");
                //System.out.println(comparision.getComparision(index));
                int line = comparision.getComparision(index);
                this.bufferOut.println("Diference at Line : " + line);
                this.bufferOut.print(program1.getName() + " : ");
                this.bufferOut.print(program1.getInstruccion(line).getInstruccionHex());
                this.bufferOut.println(" - " + program1.getInstruccion(line).getInstruccionBin());
                if(program1.getConfiguration(line).endsWith("\n"))
                    this.bufferOut.print(program1.getConfiguration(line));
                else
                    this.bufferOut.println(program1.getConfiguration(line));
                this.bufferOut.print(program2.getName() + " : ");
                this.bufferOut.print(program2.getInstruccion(line).getInstruccionHex());
                this.bufferOut.println(" - " + program2.getInstruccion(line).getInstruccionBin());
                if(program2.getConfiguration(line).endsWith("\n"))
                    this.bufferOut.print(program2.getConfiguration(line));
                else
                    this.bufferOut.println(program2.getConfiguration(line));
                this.bufferOut.println();
                }
            this.bufferOut.close();
            this.fileWriter.close();
            }
        catch (IOException ioe)
            {
            System.err.println("IOException: " + ioe.getMessage());
            }
        }

}
```

Instruccion.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Program;

import java.io.ByteArrayInputStream;

/**
```

```
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Instruccion
{

   private byte[] instruccion;
   private String instruccionBin;
   private String instruccionHex;

   public Instruccion(byte[] instruccion)
      {
      this.instruccion = new byte[4];
      this.instruccion[0] = instruccion[0];
      this.instruccion[1] = instruccion[1];
      this.instruccion[2] = instruccion[2];
      this.instruccion[3] = instruccion[3];
      this.instruccionBin = this.toBinary(this.instruccion);
      this.instruccionHex = this.toHexadecimal(this.instruccion).toUpperCase();
      }

   public byte[] getInstruccion()
      {
      return this.instruccion;
      }

   public String getInstruccionBin()
      {
      return this.instruccionBin;
      }

   public String getInstruccionHex()
      {
      return this.instruccionHex;
      }

   private String toHexadecimal(byte[] hex)
      {
            String result="";
            ByteArrayInputStream input = new ByteArrayInputStream(hex);
            String resultAux;
            int read = input.read();
            while(read != -1)
```

```
            {
                resultAux = Integer.toHexString(read);
                if(resultAux.length() < 2)
                    result += "0";
                result += resultAux;
                read = input.read();
            }
            return result.toUpperCase();
        }

    private String toBinary(byte[] hex)
        {
            String result="";
            ByteArrayInputStream input = new ByteArrayInputStream(hex);
            String resultAux;
            int read = input.read();
            while(read != -1)
            {
                resultAux = Integer.toBinaryString(read);
                while(resultAux.length() < 8)
                   resultAux = "0" + resultAux;
                result += resultAux;
                read = input.read();
            }
            return result;
        }

}
```

Program.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Program;

import java.util.ArrayList;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */
```

```
public class Program
{

   private String name;
   private ArrayList program;
   private ArrayList configuration;

   public Program(String name)
      {
      this.name = name;
      this.program = new ArrayList();
      this.configuration = new ArrayList();
      }

   public String getName()
      {
      return this.name;
      }

   public void addInstruccion(Instruccion instruccion)
      {
      this.program.add(instruccion);
      }

   public void addConfiguration(String configuration)
      {
      this.configuration.add(configuration);
      }

   public void addConfiguration(int index, String configuration)
      {
      this.configuration.add(index, configuration);
      }

   public Instruccion getInstruccion(int index)
      {
      return (Instruccion)this.program.get(index);
      }

   public String getConfiguration(int index)
      {
      return (String)this.configuration.get(index);
      }

   public int getSize()
      {
      return this.program.size();
      }
}
```

Código Virtex II Partial Reconfiguration

Main.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package virtexiipartialreconfiguration;

import IO.*;
import Program.*;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Main
{

    /**
     * @param args the command line arguments
     */

    private final static int BEGINPACKETBITS = 31;
    private final static int ENDPACKETBITS = 29;
    private final static int WRITEBIT = 28;
    private final static int READBIT = 27;
```

```
    private final static int BEGINTYPE1WORD = 10;
    private final static int ENDTYPE1WORD = 0;
    private final static int BEGINTYPE2WORD = 26;
    private final static int ENDTYPE2WORD = 0;
    private final static int BEGINCONFIGREG = 17;
    private final static int ENDCONFIGREG = 13;
    private final static int BEGINCMD = 3;
    private final static int ENDCMD = 0;
    private final static String[] READWRITE = {" "," Read ", " Write "};
    private final static String[] CONFIGREG = {"CRC","FAR","FDRI","FDRO","CMD",
        "CTL","MASK","STAT","LOUT","COR","MFWR","FLR","KEY","CBC","IDCODE"};
    private final static String[] CMD = {" ","WCFG","MFWR","LFRM","RCFG","START",
        "RCAP","RCRC","AGHIGH","SWITCH","GRESTORE","SHUTDOWN","GCAPTURE","D
ESYNCH"};
    private final static int nGCLKColumns = 1;
    private final static int nGCLKFramesPerColumn = 4;
    private final static int nIOBColumns = 2;
    private final static int nIOBFramesPerColumn = 4;
    private final static int nIOIColumns = 2;
    private final static int nIOIFramesPerColumn = 22;
    private final static int nCLBFramesPerColumn = 22;
    private final static int nBRAMFramesPerColumn = 64;
    private final static int nBRAMInterconnectFramesPerColumn = 22;


    private static Read input = null;
    private static Write output = null;
    private static Write outputRestorer = null;
    private static String inputFile = "";
    private static String outputFile = "";
    private static int nFrame = -1;
    private static int nWord = -1;
    private static int nBit = -1;
    private static int totalFrames = -1;
    private static int totalFrameLength = -1;
    private static int nCLBColumns = -1;
```

```
private static int nBRAMColumns = -1;
private static int nBRAMInterconnectColumns = -1;

public static void main(String[] args) throws Exception
   {
   // TODO code application logic here
   Main.evaluateArgs(args);
   Main.input = new Read(Main.inputFile);
   Main.output = new Write(Main.outputFile);
   Main.outputRestorer = new Write(Main.outputFile + "_Restorer");
   Program inputProgram = new Program();
   Main.output.open();
   Main.outputRestorer.open();
   Main.input.readFile(inputProgram,Main.output,Main.outputRestorer);
   Main.partialReconfigurationTranslate(inputProgram);
   Main.output.close();
   Main.outputRestorer.close();
   }

private static void evaluateArgs(String[] args) throws Exception
   {
   int i = 0;
   if (args.length < 1)
      Main.printHelp();
   while (i < args.length)
      {
      if(args[i] == null);
      else if(args[i].equals("-i"))
         {
         i++;
         Main.inputFile = args[i];
         }
      else if(args[i].equals("-o"))
         {
         i++;
```

```
            Main.outputFile = args[i];
            }
        else if(args[i].equals("-f"))
            {
            i++;
            Main.nWord = Integer.parseInt(args[i]);
            if(Main.nWord < 0)
                Main.nWord = 0;
            }
        else if(args[i].equals("-b"))
            {
            i++;
            Main.nBit = Integer.parseInt(args[i]);
            if(Main.nBit < 0)
                Main.nBit = 0;
            if(Main.nBit > 31)
                Main.nBit = 31;
            }
        else if(args[i].equals("-h"))
            {
            i++;
            Main.printHelp();
            }
        i++;
        }
    if((Main.inputFile.compareTo("") == 0) || (Main.outputFile.compareTo("") == 0))
        throw new Exception("No input/output File");
    if((Main.nWord == -1) || (Main.nBit == -1))
        throw new Exception("No number Frame/Bit");
    }

private static void printHelp()
    {
    System.out.println("Not yet implemented");
    }
```

```
private static void partialReconfigurationTranslate(Program program)
   {
   Instruccion instruccion;
   int index = 0;
   String configuration = "";
   while(index < program.getSize())
      {
      instruccion = program.getInstruccion(index);
      //configuration = "Type " + Main.evaluatePacket(instruccion) +
      //       Main.READWRITE[Main.evaluateWR(instruccion)] +
      //       Main.evaluateTypeWord(instruccion) + " Words ";
      if(instruccion.getInstruccionHex().compareTo("AA995566") == 0)
         {
         //configuration = "Sync Word";
         //Main.output.writeInstruccionByte(instruccion);
         //Main.outputRestorer.writeInstruccionByte(instruccion);
         }
      else if(instruccion.getInstruccionHex().compareTo("20000000") == 0)
         {
         int count = 1;
         while(instruccion.getInstruccionHex().compareTo("20000000") == 0)
            {
            //configuration = "Type " + Main.evaluatePacket(instruccion) +
            //       " NOOP Word " + count;
            Main.output.writeInstruccionByte(instruccion);
            Main.outputRestorer.writeInstruccionByte(instruccion);
            count++;
            index++;
            if(index < program.getSize())
               instruccion = program.getInstruccion(index);
            else
               return;
            }
         index--;
```

```
    }
else if(Main.evaluatePacket(instruccion) == 1)
    {
    if(Main.evaluateWR(instruccion) == 2)
        {
        //if(Main.evaluateConfReg(instruccion) < Main.CONFIGREG.length)
        //    configuration += "to " + Main.CONFIGREG[Main.evaluateConfReg(instruccion)];
        //else
        //    configuration += "to Not Configuration Register Found";
        switch(Main.evaluateConfReg(instruccion))
            {
            case 0 :
                Main.output.writeInstruccionByte(instruccion);
                Main.outputRestorer.writeInstruccionByte(instruccion);
                index = Main.evaluateConfigRegWriteCRC(program, instruccion, index);
                if(index == -1)
                    return;
                break;
            case 1 :
                Main.output.writeInstruccionByte(instruccion);
                Main.outputRestorer.writeInstruccionByte(instruccion);
                index = Main.evaluateConfigRegWriteFAR(program, instruccion, index);
                if(index == -1)
                    return;
                break;
            case 2 :
                Instruccion instruccion2 = new Instruccion(instruccion);
                instruccion2.setInstruccionOneFrameFDRI(Main.totalFrameLength * 2);
                Main.output.writeInstruccionByte(instruccion2);
                Main.outputRestorer.writeInstruccionByte(instruccion2);
                index = Main.evaluateConfigRegWriteFDRI(program, instruccion, index);
                if(index == -1)
                    return;
                break;
            case 3 :
```

```
        //configuration += " (FORBIDDEN)";
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteFDRO(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 4 :
        //Main.output.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteCMD(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 5 :
        //Main.output.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteCTL(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 6 :
        //Main.output.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteMASK(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 7 :
        //configuration += " (FORBIDDEN)";
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteSTAT(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 8 :
        Main.output.writeInstruccionByte(instruccion);
```

```
        Main.outputRestorer.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteLOUT(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 9 :
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteCOR(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 10 :
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteMFWR(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 11 :
        //Main.output.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteFLR(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 12 :
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        index = Main.evaluateConfigRegWriteKEY(program, instruccion, index);
        if(index == -1)
           return;
        break;
    case 13 :
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
```

```
            index = Main.evaluateConfigRegWriteCBC(program, instruccion, index);
            if(index == -1)
               return;
            break;
      case 14 :
            Main.output.writeInstruccionByte(instruccion);
            Main.outputRestorer.writeInstruccionByte(instruccion);
            index = Main.evaluateConfigRegWriteIDCODE(program, instruccion, index);
            if(index == -1)
               return;
            break;
      default : break;
      }
   }
else if(Main.evaluateWR(instruccion) == 1)
   {
   Main.output.writeInstruccionByte(instruccion);
   Main.outputRestorer.writeInstruccionByte(instruccion);
   //int confReg = Main.evaluateConfReg(instruccion);
   //if(Main.evaluateConfReg(instruccion) < Main.CONFIGREG.length)
   //    configuration += "to " + Main.CONFIGREG[Main.evaluateConfReg(instruccion)];
   //else
   //    configuration += "to Not Configuration Register Found";
   String configurationForbidden = "";
   switch(Main.evaluateConfReg(instruccion))
      {
      case 0 : break;
      case 1 : break;
      case 2 : configurationForbidden = " (FORBIDDEN)";
            break;
      case 3 : break;
      case 4 : break;
      case 5 : break;
      case 6 : break;
      case 7 : break;
```

```
    case 8 : configurationForbidden = " (FORBIDDEN)";
            break;
    case 9 : break;
    case 10 : configurationForbidden = " (FORBIDDEN)";
            break;
    case 11 : break;
    case 12 : configurationForbidden = " (FORBIDDEN)";
            break;
    case 13 : configurationForbidden = " (FORBIDDEN)";
            break;
    case 14 : break;
    default : break;
    }
if(Main.evaluateTypeWord(instruccion) == 0)
    {
    //configuration += configurationForbidden;
    index++;
    if(index < program.getSize())
       instruccion = program.getInstruccion(index);
    else
       return;
    if(Main.evaluatePacket(instruccion) != 2)
       if(Main.evaluateWR(instruccion) != 1)
          {
          index--;
          break;
          }
    //configuration = "Type " + Main.evaluatePacket(instruccion) +
    //       Main.READWRITE[Main.evaluateWR(instruccion)] +
    //       Main.evaluateTypeWord(instruccion) + " Words of " +
    //       Main.CONFIGREG[confReg] + configurationForbidden;
    Main.output.writeInstruccionByte(instruccion);
    Main.outputRestorer.writeInstruccionByte(instruccion);
    }
}
```

```
        }
      //else
      //    configuration += "Not Command Found";
      //configuration += "\n";
      index++;
      }
   }

private static int evaluatePacket(Instruccion instruccion)
   {
   int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINPACKETBITS - 1;
   int endIndex = instruccion.getInstruccionBin().length() - Main.ENDPACKETBITS;
   return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
   }

private static int evaluateTypeWord(Instruccion instruccion)
   {
   int beginIndex;
   int endIndex;
   if(Main.evaluatePacket(instruccion) == 1)
      {
      beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINTYPE1WORD - 1;
      endIndex = instruccion.getInstruccionBin().length() - Main.ENDTYPE1WORD;
      }
   else
      {
      beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINTYPE2WORD - 1;
      endIndex = instruccion.getInstruccionBin().length() - Main.ENDTYPE2WORD;
      }
   return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
   }

private static int evaluateWR(Instruccion instruccion)
   {
   int beginIndex = instruccion.getInstruccionBin().length() - Main.WRITEBIT - 1;
```

```
    int endIndex = instruccion.getInstruccionBin().length() - Main.WRITEBIT;
    if(instruccion.getInstruccionBin().substring(beginIndex,endIndex).compareTo("1") == 0)
        return 2;
    beginIndex = instruccion.getInstruccionBin().length() - Main.READBIT - 1;
    endIndex = instruccion.getInstruccionBin().length() - Main.READBIT;
    if(instruccion.getInstruccionBin().substring(beginIndex, endIndex).compareTo("1") == 0)
        return 1;
    return 0;
    }

private static int evaluateConfReg(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINCONFIGREG - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.ENDCONFIGREG;
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex,endIndex),2);
    }

private static int evaluateCMD(Instruccion instruccion)
    {
    int beginIndex = instruccion.getInstruccionBin().length() - Main.BEGINCMD - 1;
    int endIndex = instruccion.getInstruccionBin().length() - Main.ENDCMD;
    return Integer.parseInt(instruccion.getInstruccionBin().substring(beginIndex, endIndex), 2);
    }

private static String evaluateID(Instruccion instruccion)
    {
    String device = "No Virtex-II Found";
    if(instruccion.getInstruccionHex().compareTo("01226093") == 0)
        {
        device = "XC2VP2";
        Main.totalFrames = 884;
        Main.totalFrameLength = 46;
        Main.nCLBColumns = 22;
        Main.nBRAMColumns = 4;
        Main.nBRAMInterconnectColumns = 4;
```

```
    }
if(instruccion.getInstruccionHex().compareTo("0123E093") == 0)
    {
    device = "XC2VP4";
    Main.totalFrames = 884;
    Main.totalFrameLength = 106;
    Main.nCLBColumns = 22;
    Main.nBRAMColumns = 4;
    Main.nBRAMInterconnectColumns = 4;
    }
if(instruccion.getInstruccionHex().compareTo("0124A093") == 0)
    {
    device = "XC2VP7";
    Main.totalFrames = 1320;
    Main.totalFrameLength = 106;
    Main.nCLBColumns = 34;
    Main.nBRAMColumns = 6;
    Main.nBRAMInterconnectColumns = 6;
    }
if(instruccion.getInstruccionHex().compareTo("01266093") == 0)
    {
    device = "XC2VP20";
    Main.totalFrames = 1756;
    Main.totalFrameLength = 146;
    Main.nCLBColumns = 46;
    Main.nBRAMColumns = 8;
    Main.nBRAMInterconnectColumns = 8;
    }
if(instruccion.getInstruccionHex().compareTo("01866093") == 0)
    {
    device = "XC2VPX20";
    Main.totalFrames = 1756;
    Main.totalFrameLength = 146;
    Main.nCLBColumns = 46;
    Main.nBRAMColumns = 8;
```

```
    Main.nBRAMInterconnectColumns = 8;
    }
if(instruccion.getInstruccionHex().compareTo("0127E093") == 0)
    {
    device = "XC2VP30";
    Main.totalFrames = 1756;
    Main.totalFrameLength = 206;
    Main.nCLBColumns = 46;
    Main.nBRAMColumns = 8;
    Main.nBRAMInterconnectColumns = 8;
    }
if(instruccion.getInstruccionHex().compareTo("01292093") == 0)
    {
    device = "XC2VP40";
    Main.totalFrames = 2192;
    Main.totalFrameLength = 226;
    Main.nCLBColumns = 58;
    Main.nBRAMColumns = 10;
    Main.nBRAMInterconnectColumns = 10;
    }
if(instruccion.getInstruccionHex().compareTo("0129E093") == 0)
    {
    device = "XC2VP50";
    Main.totalFrames = 2628;
    Main.totalFrameLength = 226;
    Main.nCLBColumns = 70;
    Main.nBRAMColumns = 12;
    Main.nBRAMInterconnectColumns = 12;
    }
if(instruccion.getInstruccionHex().compareTo("012BA093") == 0)
    {
    device = "XC2VP70";
    Main.totalFrames = 3064;
    Main.totalFrameLength = 266;
    Main.nCLBColumns = 82;
```

```
        Main.nBRAMColumns = 14;
        Main.nBRAMInterconnectColumns = 14;
        }
    if(instruccion.getInstruccionHex().compareTo("018BA093") == 0)
        {
        device = "XC2VPX70";
        Main.totalFrames = 3064;
        Main.totalFrameLength = 266;
        Main.nCLBColumns = 82;
        Main.nBRAMColumns = 14;
        Main.nBRAMInterconnectColumns = 14;
        }
    if(instruccion.getInstruccionHex().compareTo("012D6093") == 0)
        {
        device = "XC2VP100";
        Main.totalFrames = 3500;
        Main.totalFrameLength = 306;
        Main.nCLBColumns = 94;
        Main.nBRAMColumns = 16;
        Main.nBRAMInterconnectColumns = 16;
        }
    if(Main.nWord > (Main.totalFrames * Main.totalFrameLength))
        Main.nWord = Main.totalFrames * Main.totalFrameLength;
    return device;
    }

  private static int evaluateConfigRegWriteCRC(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
```

```
        else
            return -1;
        if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
                return index--;
        //configuration = "Type " + Main.evaluatePacket(instruccion) +
        //        Main.READWRITE[Main.evaluateWR(instruccion)] +
        //        Main.evaluateTypeWord(instruccion) + " Words to CRC";
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        instruccion.setInstruccionNoCRC();
        //configuration = "Packet Data: Word " + i + " CRC Register = 0x" +
        //        instruccion.getInstruccionHex();
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        }
    return index;
    }


  private static int evaluateConfigRegWriteFAR(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
```

```
    if(index < program.getSize())
       instruccion = program.getInstruccion(index);
    else
       return -1;
    if(Main.evaluatePacket(instruccion) != 2)
       if(Main.evaluateWR(instruccion) != 2)
          return index--;
    //configuration = "Type " + Main.evaluatePacket(instruccion) +
    //       Main.READWRITE[Main.evaluateWR(instruccion)] +
    //       Main.evaluateTypeWord(instruccion) + " Words to FAR";
    Main.output.writeInstruccionByte(instruccion);
    Main.outputRestorer.writeInstruccionByte(instruccion);
    }
int numWords = Main.evaluateTypeWord(instruccion);
for(int i = 1 ; i <= numWords ; i++)
    {
    index++;
    if(index < program.getSize())
       instruccion = program.getInstruccion(index);
    else
       return -1;
    //configuration = "Packet Data: Word " + i + " Frame Address Register = 0x" +
    //       instruccion.getInstruccionHex();
    Main.nFrame = (Main.nWord - 1) / Main.totalFrameLength;
    int BA = 0;
    int MJA = 0;
    int MNA = Main.nFrame - (Main.nGCLKColumns * Main.nGCLKFramesPerColumn);
    if(MNA < 0)
       {
       MNA = MNA + (Main.nGCLKColumns * Main.nGCLKFramesPerColumn);
       }
    else
       {
       MNA = MNA - (Main.nIOBColumns/2 * Main.nIOBFramesPerColumn);
       if(MNA < 0)
```

```
    {
    MJA = 1;
    MNA = MNA + (Main.nIOBColumns/2 * Main.nIOBFramesPerColumn);
    }
  else
    {
    MNA = MNA - (Main.nIOIColumns/2 * Main.nIOIFramesPerColumn);
    if(MNA < 0)
      {
      MJA = 2;
      MNA = MNA + (Main.nIOIColumns/2 * Main.nIOIFramesPerColumn);
      }
    else
      {
      if(MNA / (Main.nCLBColumns * Main.nCLBFramesPerColumn) == 0)
        {
        MJA = (MNA / Main.nCLBFramesPerColumn) + Main.nGCLKColumns +
              Main.nIOBColumns/2 + Main.nIOIColumns/2;
        MNA = MNA % Main.nCLBFramesPerColumn;
        }
      else
        {
        MNA = MNA - (Main.nCLBColumns * Main.nCLBFramesPerColumn) -
              (Main.nIOIColumns/2 * Main.nIOIFramesPerColumn);
        MJA = Main.nGCLKColumns + Main.nIOBColumns/2 + Main.nIOIColumns/2 +
              Main.nCLBColumns;
        if(MNA < 0)
          MNA = MNA + (Main.nIOIColumns/2 * Main.nIOIFramesPerColumn);
        else
          {
          MNA = MNA - (Main.nIOBColumns/2 * Main.nIOBFramesPerColumn);
          MJA = MJA + 1;
          if(MNA < 0)
            MNA = MNA + (Main.nIOBColumns/2 * Main.nIOBFramesPerColumn);
          else
```

```
                        {
                        BA = 1;
                        MJA = 0;
                        if(MNA / (Main.nBRAMColumns * Main.nBRAMFramesPerColumn) == 0)
                           {
                           MJA = MNA / Main.nBRAMFramesPerColumn;
                           MNA = MNA % Main.nBRAMFramesPerColumn;
                           }
                        else
                           {
                           BA = 2;
                           MNA = MNA - (Main.nBRAMColumns *
Main.nBRAMFramesPerColumn);
                           MJA = MNA / Main.nBRAMInterconnectFramesPerColumn;
                           MNA = MNA % Main.nBRAMInterconnectFramesPerColumn;
                           }
                        }
                     }
                  }
               }
            }
         }
         instruccion.setInstruccionFrame(BA,MJA,MNA);
         Main.output.writeInstruccionByte(instruccion);
         Main.outputRestorer.writeInstruccionByte(instruccion);
         }
      return index;
      }


   private static int evaluateConfigRegWriteFDRI(Program program, Instruccion instruccion, int
index)
      {
      String configuration;
      if(Main.evaluateTypeWord(instruccion) == 0)
         {
         index++;
```

```
   if(index < program.getSize())
      instruccion = program.getInstruccion(index);
   else
      return -1;
   if(Main.evaluatePacket(instruccion) != 2)
      if(Main.evaluateWR(instruccion) != 2)
         return index--;
   //configuration = "Type " + Main.evaluatePacket(instruccion) +
   //       Main.READWRITE[Main.evaluateWR(instruccion)] +
   //       Main.evaluateTypeWord(instruccion) + " Words to FDRI";
   //Main.output.writeInstruccionByte(instruccion);
   }
int numWords = Main.evaluateTypeWord(instruccion);
int iniWord = Main.nFrame * Main.totalFrameLength;
int auxWord = 0;
for(int i = 1 ; i <= numWords ; i++)
   {
   index++;
   if(index < program.getSize())
      instruccion = program.getInstruccion(index);
   else
      return -1;
   //configuration = "Packet Data: Word " + i + " Frame Data Input Register = 0x" +
   //       instruccion.getInstruccionHex();
   auxWord = (i - 1) - iniWord;
   if(((0 <= auxWord) && (auxWord < Main.totalFrameLength)) ||
         (Main.totalFrames * Main.totalFrameLength < i))
      {
      Main.outputRestorer.writeInstruccionByte(instruccion);
      if(Main.nWord == i)
         instruccion.setInstruccionBitError(Main.nBit);
      Main.output.writeInstruccionByte(instruccion);
      }
   }
index++;
```

```
    if(index < program.getSize())
       instruccion = program.getInstruccion(index);
    else
       return -1;
    instruccion.setInstruccionNoCRC();
    //configuration = "Packet Data: AutoCRC Word = 0x" + instruccion.getInstruccionHex();
    Main.output.writeInstruccionByte(instruccion);
    Main.outputRestorer.writeInstruccionByte(instruccion);
    return index;
    }

  private static int evaluateConfigRegWriteFDRO(Program program, Instruccion instruccion, int
index)
    {
    String configuration;


    return index;
    }


  private static int evaluateConfigRegWriteCMD(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    Instruccion instruccion2;
    if(Main.evaluateTypeWord(instruccion) == 0)
       {
       index++;
       if(index < program.getSize())
          instruccion = program.getInstruccion(index);
       else
          return -1;
       if(Main.evaluatePacket(instruccion) != 2)
          if(Main.evaluateWR(instruccion) != 2)
             return index--;
       //configuration = "Type " + Main.evaluatePacket(instruccion) +
       //       Main.READWRITE[Main.evaluateWR(instruccion)] +
```

```
        //      Main.evaluateTypeWord(instruccion) + " Words to CMD";
          Main.output.writeInstruccionByte(instruccion);
          Main.outputRestorer.writeInstruccionByte(instruccion);
          }
       int numWords = Main.evaluateTypeWord(instruccion);
       for(int i = 1 ; i <= numWords ; i++)
          {
          index++;
          if(index < program.getSize())
             instruccion2 = program.getInstruccion(index);
          else
             return -1;
          if(Main.evaluateCMD(instruccion2) < Main.CMD.length)
             configuration = "Packet Data: Word " + i + " CMD Register = " +
                   Main.CMD[Main.evaluateCMD(instruccion2)] + " Command";
          else
             return index--;
          if((Main.evaluateCMD(instruccion2) != 9) && (Main.evaluateCMD(instruccion2) != 10)
&& (Main.evaluateCMD(instruccion2) != 5))
             {
             Main.output.writeInstruccionByte(instruccion);
             Main.outputRestorer.writeInstruccionByte(instruccion);
             Main.output.writeInstruccionByte(instruccion2);
             Main.outputRestorer.writeInstruccionByte(instruccion2);
             }
          }
       return index;
       }


    private static int evaluateConfigRegWriteCTL(Program program, Instruccion instruccion, int
index)
       {
       String configuration;
       if(Main.evaluateTypeWord(instruccion) == 0)
          {
          index++;
```

```
    if(index < program.getSize())
       instruccion = program.getInstruccion(index);
    else
       return -1;
    if(Main.evaluatePacket(instruccion) != 2)
       if(Main.evaluateWR(instruccion) != 2)
          return index--;
    //configuration = "Type " + Main.evaluatePacket(instruccion) +
    //       Main.READWRITE[Main.evaluateWR(instruccion)] +
    //       Main.evaluateTypeWord(instruccion) + " Words to CTL";
    //Main.output.writeInstruccionByte(instruccion);
    }
  int numWords = Main.evaluateTypeWord(instruccion);
  for(int i = 1 ; i <= numWords ; i++)
    {
    index++;
    if(index < program.getSize())
       instruccion = program.getInstruccion(index);
    else
       return -1;
    //configuration = "Packet Data: Word " + i + " Control Register = 0x" +
    //       instruccion.getInstruccionHex();
    //Main.output.writeInstruccionByte(instruccion);
    }
  return index;
  }

 private static int evaluateConfigRegWriteMASK(Program program, Instruccion instruccion, int
index)
  {
  String configuration;
  if(Main.evaluateTypeWord(instruccion) == 0)
    {
    index++;
    if(index < program.getSize())
```

```
            instruccion = program.getInstruccion(index);
        else
            return -1;
        if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
                return index--;
        //configuration = "Type " + Main.evaluatePacket(instruccion) +
        //        Main.READWRITE[Main.evaluateWR(instruccion)] +
        //        Main.evaluateTypeWord(instruccion) + " Words to MASK";
        //Main.output.writeInstruccionByte(instruccion);
        }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        //configuration = "Packet Data: Word " + i + " Mask Register = 0x" +
        //        instruccion.getInstruccionHex();
        //Main.output.writeInstruccionByte(instruccion);
        }
    return index;
    }

  private static int evaluateConfigRegWriteSTAT(Program program, Instruccion instruccion, int
index)
    {
    String configuration;

    return index;
    }

  private static int evaluateConfigRegWriteLOUT(Program program, Instruccion instruccion, int
index)
```

```
{
String configuration;
if(Main.evaluateTypeWord(instruccion) == 0)
   {
   index++;
   if(index < program.getSize())
      instruccion = program.getInstruccion(index);
   else
      return -1;
   if(Main.evaluatePacket(instruccion) != 2)
      if(Main.evaluateWR(instruccion) != 2)
         return index--;
   //configuration = "Type " + Main.evaluatePacket(instruccion) +
   //       Main.READWRITE[Main.evaluateWR(instruccion)] +
   //       Main.evaluateTypeWord(instruccion) + " Words to LOUT";
   Main.output.writeInstruccionByte(instruccion);
   Main.outputRestorer.writeInstruccionByte(instruccion);
   }
int numWords = Main.evaluateTypeWord(instruccion);
for(int i = 1 ; i <= numWords ; i++)
   {
   index++;
   if(index < program.getSize())
      instruccion = program.getInstruccion(index);
   else
      return -1;
   //configuration = "Packet Data: Word " + i + " Legacy Output Register = 0x" +
   //       instruccion.getInstruccionHex();
   Main.output.writeInstruccionByte(instruccion);
   Main.outputRestorer.writeInstruccionByte(instruccion);
   }
return index;
}
```

```
private static int evaluateConfigRegWriteCOR(Program program, Instruccion instruccion, int
```

```
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      if(Main.evaluatePacket(instruccion) != 2)
        if(Main.evaluateWR(instruccion) != 2)
          return index--;
      //configuration = "Type " + Main.evaluatePacket(instruccion) +
      //       Main.READWRITE[Main.evaluateWR(instruccion)] +
      //       Main.evaluateTypeWord(instruccion) + " Words to COR";
      Main.output.writeInstruccionByte(instruccion);
      Main.outputRestorer.writeInstruccionByte(instruccion);
      }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      //configuration = "Packet Data: Word " + i + " Configuration Options Register = 0x" +
      //       instruccion.getInstruccionHex();
      instruccion.setInstruccionBit(29);
      Main.output.writeInstruccionByte(instruccion);
      Main.outputRestorer.writeInstruccionByte(instruccion);
      }
    return index;
    }
```

```
    private static int evaluateConfigRegWriteMFWR(Program program, Instruccion instruccion, int
index)
        {
        String configuration;
        if(Main.evaluateTypeWord(instruccion) == 0)
            {
            index++;
            if(index < program.getSize())
                instruccion = program.getInstruccion(index);
            else
                return -1;
            if(Main.evaluatePacket(instruccion) != 2)
                if(Main.evaluateWR(instruccion) != 2)
                    return index--;
            //configuration = "Type " + Main.evaluatePacket(instruccion) +
            //        Main.READWRITE[Main.evaluateWR(instruccion)] +
            //        Main.evaluateTypeWord(instruccion) + " Words to MFWR";
            Main.output.writeInstruccionByte(instruccion);
            Main.outputRestorer.writeInstruccionByte(instruccion);
            }
        int numWords = Main.evaluateTypeWord(instruccion);
        for(int i = 1 ; i <= numWords ; i++)
            {
            index++;
            if(index < program.getSize())
                instruccion = program.getInstruccion(index);
            else
                return -1;
            //configuration = "Packet Data: Word " + i + " Multiple Frame Write Register = 0x" +
            //        instruccion.getInstruccionHex();
            Main.output.writeInstruccionByte(instruccion);
            Main.outputRestorer.writeInstruccionByte(instruccion);
            }
        return index;
```

```
    }

    private static int evaluateConfigRegWriteFLR(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
       {
       index++;
       if(index < program.getSize())
          instruccion = program.getInstruccion(index);
       else
          return -1;
       if(Main.evaluatePacket(instruccion) != 2)
          if(Main.evaluateWR(instruccion) != 2)
             return index--;
       //configuration = "Type " + Main.evaluatePacket(instruccion) +
       //       Main.READWRITE[Main.evaluateWR(instruccion)] +
       //       Main.evaluateTypeWord(instruccion) + " Words to FLR";
       //Main.output.writeInstruccionByte(instruccion);
       }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
       {
       index++;
       if(index < program.getSize())
          instruccion = program.getInstruccion(index);
       else
          return -1;
       //configuration = "Packet Data: Word " + i + " Frame Length Register = " +
       //       Integer.parseInt(instruccion.getInstruccionBin(),2) +
       //       " Words = 0x" + instruccion.getInstruccionHex();
       //Main.output.writeInstruccionByte(instruccion);
       }
    return index;
```

```
    }

  private static int evaluateConfigRegWriteKEY(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      if(Main.evaluatePacket(instruccion) != 2)
        if(Main.evaluateWR(instruccion) != 2)
          return index--;
      //configuration = "Type " + Main.evaluatePacket(instruccion) +
      //       Main.READWRITE[Main.evaluateWR(instruccion)] +
      //       Main.evaluateTypeWord(instruccion) + " Words to KEY";
      Main.output.writeInstruccionByte(instruccion);
      Main.outputRestorer.writeInstruccionByte(instruccion);
      }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      //configuration = "Packet Data: Word " + i + " Initial Key Address Register = 0x" +
      //       instruccion.getInstruccionHex();
      Main.output.writeInstruccionByte(instruccion);
      Main.outputRestorer.writeInstruccionByte(instruccion);
      }
```

```
    return index;
    }

  private static int evaluateConfigRegWriteCBC(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      if(Main.evaluatePacket(instruccion) != 2)
        if(Main.evaluateWR(instruccion) != 2)
          return index--;
      //configuration = "Type " + Main.evaluatePacket(instruccion) +
      //       Main.READWRITE[Main.evaluateWR(instruccion)] +
      //       Main.evaluateTypeWord(instruccion) + " Words to CBC";
      Main.output.writeInstruccionByte(instruccion);
      Main.outputRestorer.writeInstruccionByte(instruccion);
      }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
      {
      index++;
      if(index < program.getSize())
        instruccion = program.getInstruccion(index);
      else
        return -1;
      //configuration = "Packet Data: Word " + i + " Cipher Block Chaining Register = 0x" +
      //       instruccion.getInstruccionHex();
      Main.output.writeInstruccionByte(instruccion);
      Main.outputRestorer.writeInstruccionByte(instruccion);
```

```
        }
    return index;
    }

  private static int evaluateConfigRegWriteIDCODE(Program program, Instruccion instruccion, int
index)
    {
    String configuration;
    if(Main.evaluateTypeWord(instruccion) == 0)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        if(Main.evaluatePacket(instruccion) != 2)
            if(Main.evaluateWR(instruccion) != 2)
                return index--;
        //configuration = "Type " + Main.evaluatePacket(instruccion) +
        //        Main.READWRITE[Main.evaluateWR(instruccion)] +
        //        Main.evaluateTypeWord(instruccion) + " Words to IDCODE";
        Main.output.writeInstruccionByte(instruccion);
        Main.outputRestorer.writeInstruccionByte(instruccion);
        }
    int numWords = Main.evaluateTypeWord(instruccion);
    for(int i = 1 ; i <= numWords ; i++)
        {
        index++;
        if(index < program.getSize())
            instruccion = program.getInstruccion(index);
        else
            return -1;
        //configuration = "Packet Data: Word " + i + " Device ID = 0x" +
        //        instruccion.getInstruccionHex() + " (" + Main.evaluateID(instruccion) + ")";
        Main.evaluateID(instruccion);
```

```
            Main.output.writeInstruccionByte(instruccion);
            Main.outputRestorer.writeInstruccionByte(instruccion);
            }
        return index;
        }

}
```

Read.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package IO;

import Program.*;
import java.io.*;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Read
{

   private String inputFileName;
```

```
public Read (String inputFileName)
   {
      this.inputFileName = inputFileName;
   }

public void readFile(Program program, Write w, Write wR)
   {
   FileInputStream fis = null;
   DataInputStream dis = null;
   String hexRubish;
   byte[] hex = new byte[4];
   try
      {
      fis = new FileInputStream(this.inputFileName);
      boolean rubish = true;
      dis = new DataInputStream(fis);
      short b = (short) dis.readUnsignedByte();
      w.writeByte((byte)b);
      wR.writeByte((byte)b);
      hexRubish = Integer.toHexString(b);
      while(rubish)
         {
         while(0 != hexRubish.compareTo("aa"))
            {
            b = (short) dis.readUnsignedByte();
            w.writeByte((byte)b);
            wR.writeByte((byte)b);
            hexRubish = Integer.toHexString(b);
            }
         b = (short) dis.readUnsignedByte();
         w.writeByte((byte)b);
         wR.writeByte((byte)b);
         if(0 == ((hexRubish = Integer.toHexString(b)).compareTo("99")))
            {
            b = (short) dis.readUnsignedByte();
```

```
            w.writeByte((byte)b);
            wR.writeByte((byte)b);
            if(0 == ((hexRubish = Integer.toHexString(b)).compareTo("55")))
               {
               b = (short) dis.readUnsignedByte();
               w.writeByte((byte)b);
               wR.writeByte((byte)b);
               if(0 == ((hexRubish = Integer.toHexString(b)).compareTo("66")))
                  rubish = false;
               }
            }
         }
      hex = new byte[]{(byte)0xAA, (byte)0x99, (byte)0x55, (byte)0x66};
      Instruccion instruccion = new Instruccion(hex);
      program.addInstruccion(instruccion);
      while(true)
         {
         if(dis.read(hex,0,4) == -1)
            throw new EOFException();
         instruccion = new Instruccion(hex);
         program.addInstruccion(instruccion);
         }
      }
   catch(EOFException eofe)
      {
      try
         {
         dis.close();
         fis.close();
         }
      catch(IOException ioe)
         {
         System.err.println("IOException: " + ioe.getMessage());
         }
      }
```

```
        catch(IOException ioe)
            {
            System.err.println("IOException: " + ioe.getMessage());
            }
        }

}
```

Write.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package IO;

import Program.*;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */
```

```
public class Write
{

   private String outputFileName;
   private FileOutputStream fos;
   private DataOutputStream dos;

   public Write (String outputFileName)
      {
      this.outputFileName = outputFileName + ".bit";
      }

   public void open()
      {
      try
         {
         this.fos = new FileOutputStream(this.outputFileName);
         this.dos = new DataOutputStream(fos);
         }
      catch (IOException ioe)
         {
         System.err.println("IOException: " + ioe.getMessage());
         }
      }

   public void close()
      {
      try
         {
         this.dos.close();
         this.fos.close();
         }
      catch (IOException ioe)
         {
```

```
        System.err.println("IOException: " + ioe.getMessage());
        }
    }

public void writeProgram(Program program)
    {
    try
        {
        FileWriter fW = new FileWriter(this.outputFileName + ".v2cv");
        PrintWriter bO = new PrintWriter(fW);
        for(int index = 0 ; index < program.getSize() ; index++)
            {
            bO.print(program.getInstruccion(index).getInstruccionHex());
            bO.println(" - " + program.getInstruccion(index).getInstruccionBin());
            bO.println(program.getConfiguration(index));
            }
        bO.close();
        fW.close();
        }
    catch (IOException ioe)
        {
        System.err.println("IOException: " + ioe.getMessage());
        }
    }

public void writeByte(byte b)
        {
    try {
        this.dos.write(new byte[]{(byte) b});
        }
    catch (IOException ex)
        {
        Logger.getLogger(Write.class.getName()).log(Level.SEVERE, null, ex);
        }
        }
```

```
    public void writeInstruccionByte(Instruccion instruccion)
        {
        try
            {
            this.dos.write(instruccion.getInstruccion());
            }
        catch (IOException ex)
            {
            Logger.getLogger(Write.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

}
```

Instruccion.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Program;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
```

```
 */

public class Instruccion
{

   private byte[] instruccion;
   private String instruccionBin;
   private String instruccionHex;

   public Instruccion(byte[] instruccion)
      {
      this.instruccion = new byte[4];
      this.instruccion[0] = instruccion[0];
      this.instruccion[1] = instruccion[1];
      this.instruccion[2] = instruccion[2];
      this.instruccion[3] = instruccion[3];
      this.instruccionBin = this.toBinary(this.instruccion);
      this.instruccionHex = this.toHexadecimal(this.instruccion).toUpperCase();
      }

   public Instruccion(Instruccion instruccion)
      {
      this.instruccion = new byte[4];
      this.instruccion[0] = instruccion.getInstruccion()[0];
      this.instruccion[1] = instruccion.getInstruccion()[1];
      this.instruccion[2] = instruccion.getInstruccion()[2];
      this.instruccion[3] = instruccion.getInstruccion()[3];
      this.instruccionBin = instruccion.getInstruccionBin();
      this.instruccionHex = instruccion.getInstruccionHex();
      }

   public byte[] getInstruccion()
      {
      return this.instruccion;
      }
```

```
    public String getInstruccionBin()
        {
        return this.instruccionBin;
        }

    public String getInstruccionHex()
        {
        return this.instruccionHex;
        }

    public void setInstruccionFrame(int BA, int MJA, int MNA)
        {
        this.instruccionBin = "00000" + toBinary(BA).substring(30) + toBinary(MJA).substring(24) +
              toBinary(MNA).substring(24) + "000000000";
        this.instruccionHex = this.toHexadecimal(this.instruccionBin);
        this.instruccion = this.toBytes(this.instruccionBin);
        }

    public void setInstruccionBit(int nbit)
        {
        nbit = 31 - nbit;
        if(nbit < 31)
           this.instruccionBin = this.instruccionBin.substring(0, nbit) + "1" +
this.instruccionBin.substring(nbit + 1);
        else
           this.instruccionBin = this.instruccionBin.substring(0, nbit) + "1";
        this.instruccionHex = this.toHexadecimal(this.instruccionBin);
        this.instruccion = this.toBytes(this.instruccionBin);
        }

    public void setInstruccionBitError(int nbit)
        {
        nbit = 31 - nbit;
        int bit = (Integer.valueOf(this.instruccionBin.substring(nbit, nbit + 1)) +1 ) % 2;
```

```
        if(nbit < 31)
            this.instruccionBin = this.instruccionBin.substring(0, nbit) + bit +
this.instruccionBin.substring(nbit + 1);
        else
            this.instruccionBin = this.instruccionBin.substring(0, nbit) + bit;
        this.instruccionHex = this.toHexadecimal(this.instruccionBin);
        this.instruccion = this.toBytes(this.instruccionBin);
        }

    public void setInstruccionNoCRC()
        {
        this.instruccion[0] = (byte)0x00;
        this.instruccion[1] = (byte)0x00;
        this.instruccion[2] = (byte)0xDE;
        this.instruccion[3] = (byte)0xFC;
        this.instruccionBin = "00000000000000001101111011111100";
        this.instruccionHex = "0000DEFC";
        }

    public void setInstruccionOneFrameFDRI(int nWords)
        {
        this.instruccionBin = this.instruccionBin.substring(0, 21) + toBinary(nWords).substring(21);
        this.instruccionHex = this.toHexadecimal(this.instruccionBin);
        this.instruccion = this.toBytes(this.instruccionBin);
        }

    private String toHexadecimal(byte[] hex)
        {
        String result="";
        ByteArrayInputStream input = new ByteArrayInputStream(hex);
        String resultAux;
        int read = input.read();
        while(read != -1)
            {
            resultAux = Integer.toHexString(read);
```

```
        if(resultAux.length() < 2)
            result += "0";
        result += resultAux;
        read = input.read();
        }
    return result.toUpperCase();
    }

private String toHexadecimal(int i, int size)
    {
    String result = Integer.toHexString(i);
    while(result.length() < size)
        result = "0" + result;
    return result;
    }

private String toHexadecimal(String iBin)
    {
    String result = "";
    for(int i = 0 ; i < 8 ; i++)
        {
        result = result + this.toHexadecimal(Integer.valueOf(iBin.substring(i * 4, (i * 4) + 4),2),1);
        }
    return result;
    }

private String toBinary(byte[] hex)
    {
    String result="";
    ByteArrayInputStream input = new ByteArrayInputStream(hex);
    String resultAux;
    int read = input.read();
    while(read != -1)
        {
        resultAux = Integer.toBinaryString(read);
```

```
            while(resultAux.length() < 8)
                resultAux = "0" + resultAux;
            result += resultAux;
            read = input.read();
            }
        return result;
        }

    private String toBinary(int i)
        {
        String result = Integer.toBinaryString(i);
        while(result.length() < 32)
            result = "0" + result;
        return result;
        }

    private byte[] toBytes(String iBin)
        {
        byte[] result = new byte[4];
        ByteArrayOutputStream output = new ByteArrayOutputStream();
        for (int i = 0; i < 4; i++)
            {
            if(iBin.charAt(i*8) == '1')
                output.write(Integer.valueOf(iBin.substring((i*8) + 1,(i+1) * 8),2) + 128);
            else
                output.write(Integer.valueOf(iBin.substring(i * 8 ,(i+1) * 8),2));
            }
        result = output.toByteArray();
        return result;
        }

}
```

Program.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package Program;

import java.util.ArrayList;

/**
 *
 * @author Victor Alaminos
 *
 * Supervised by Hortensia Mecha
 * Departamento Arquitectura de Computadores Y Automatica
 * Facultad de Informatica
 * Universidad Complutense de Madrid
 *
 */

public class Program
{

   private ArrayList program;
   private ArrayList configuration;

   public Program()
      {
      this.program = new ArrayList();
      this.configuration = new ArrayList();
      }

   public void addInstruccion(Instruccion instruccion)
      {
```

```
        this.program.add(instruccion);
        }

    public void addConfiguration(String configuration)
        {
        this.configuration.add(configuration);
        }

    public void addConfiguration(int index, String configuration)
        {
        this.configuration.add(index, configuration);
        }

    public Instruccion getInstruccion(int index)
        {
        return (Instruccion)this.program.get(index);
        }

    public String getConfiguration(int index)
        {
        return (String)this.configuration.get(index);
        }

    public int getSize()
        {
        return this.program.size();
        }

}
```

Apéndice 3: Nessy

En esta sección, buscamos dar una guía al usuario para facilitar el manejo de la aplicación y su correcto uso. Comenzaremos hablando de los requisitos del sistema necesarios para que se pueda ejecutar con éxito.

Requisitos del Sistema

Requisitos software

- Windows XP (la aplicación no se ejecuta para sistemas basados en 64 bits)
- La aplicación Xilinx ISE debe estar instalada
- La aplicación Impact debe estar instalada

Requisitos hardware

- Máquina con puerto serie (RS232) habilitado
- Máquina con puerto paralelo (Parallel Cable IV) habilitado
- Máquina con puerto PS2
- FPGA Virtex-II Pro (REV 04)

Además, Para ejecutar nuestra aplicación es necesaria la estructura de carpetas y archivos que se detalla en la Figura Ap3.1.

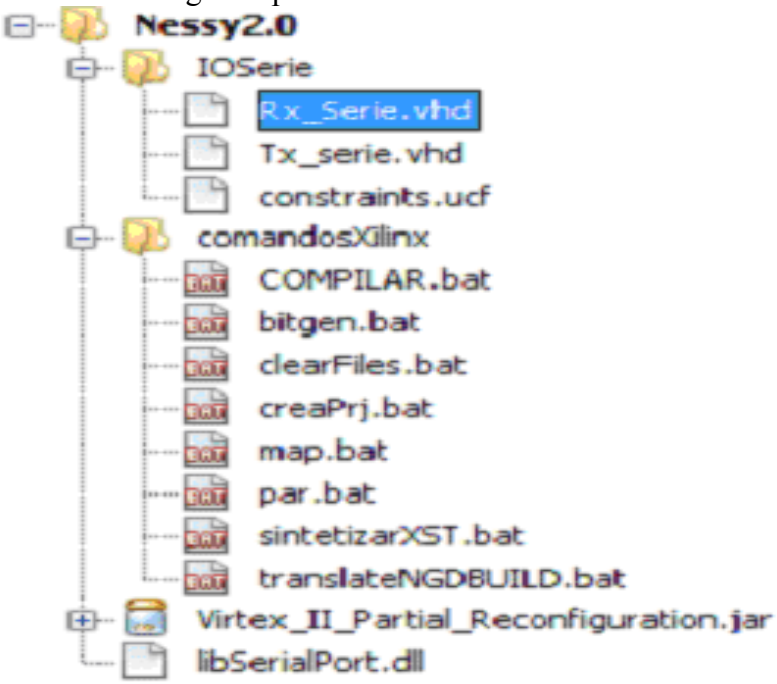

Figura Ap3.1: Directorios necesarios para la ejecución de la aplicación.

Es la ventana principal de la aplicación (ver Figura Ap3.2). Desde ella se pueden realizar todas las acciones permitidas en cada momento por la aplicación. También se visualizan distintas salidas de nuestra ejecución en cada una de las pestañas disponibles.

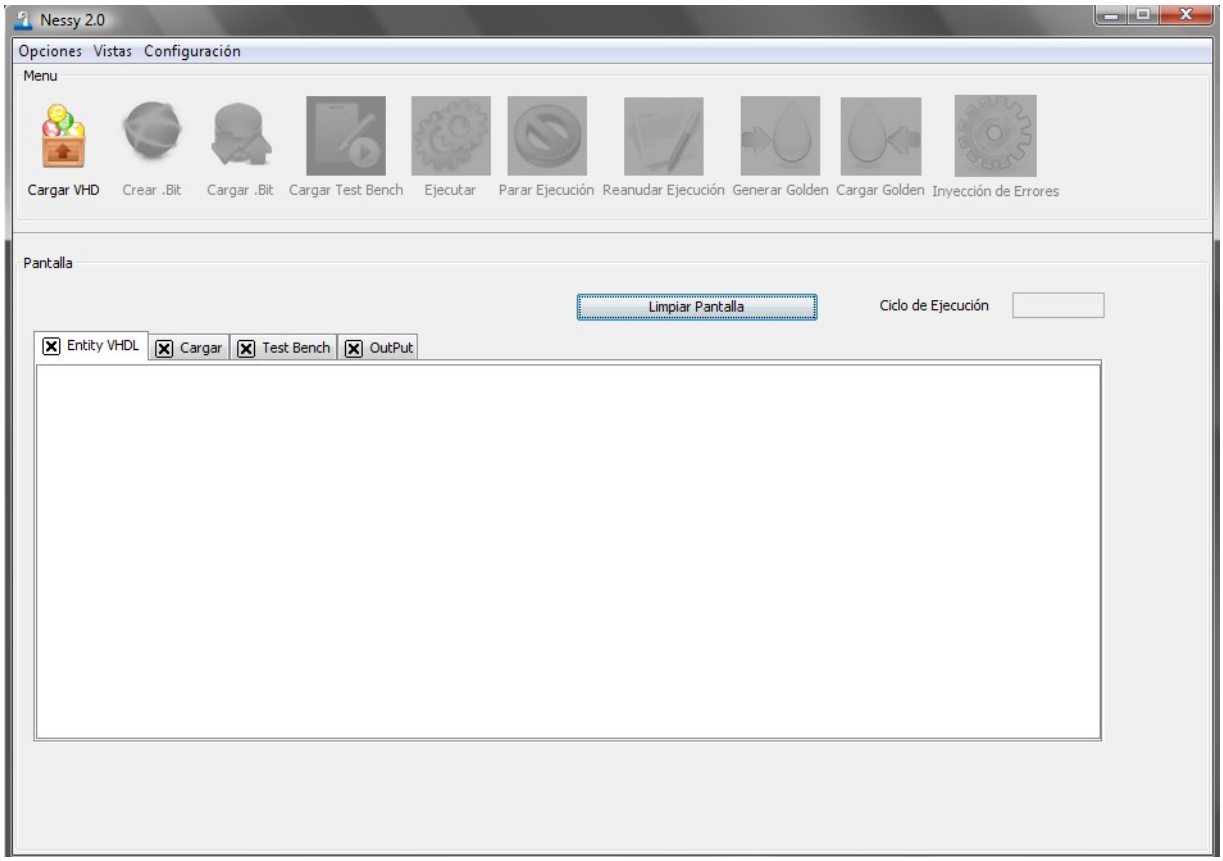

Figura Ap3.2: Ventana principal de la aplicación.

Barra de menús

Existen tres elementos desplegables en el menú de la parte superior que son el desplegable opciones, el desplegable vistas y el desplegable configuración.

Desplegable Opciones

Se muestra un menú con que contiene las mismas opciones que hay en la botonera (ver Figura Ap3.3). Si un elemento de la barra de botones se encuentra deshabilitado, en este menú aparece de forma análoga.

El efecto de pulsar cualquier acción desde el menú de opciones o desde la barra de botones tiene el mismo comportamiento.

Las opciones que se permiten realizar en este desplegable son cargar VHD, crear .bit, cargar.bit, cargar Test Bench, ejecutar, parar ejecución, reanudar ejecución, generar golden, cargar golden e inyección de errores. Más tarde, cuando expliquemos en detalle la barra de botones, detallaremos el efecto de cada elemento de este desplegable.

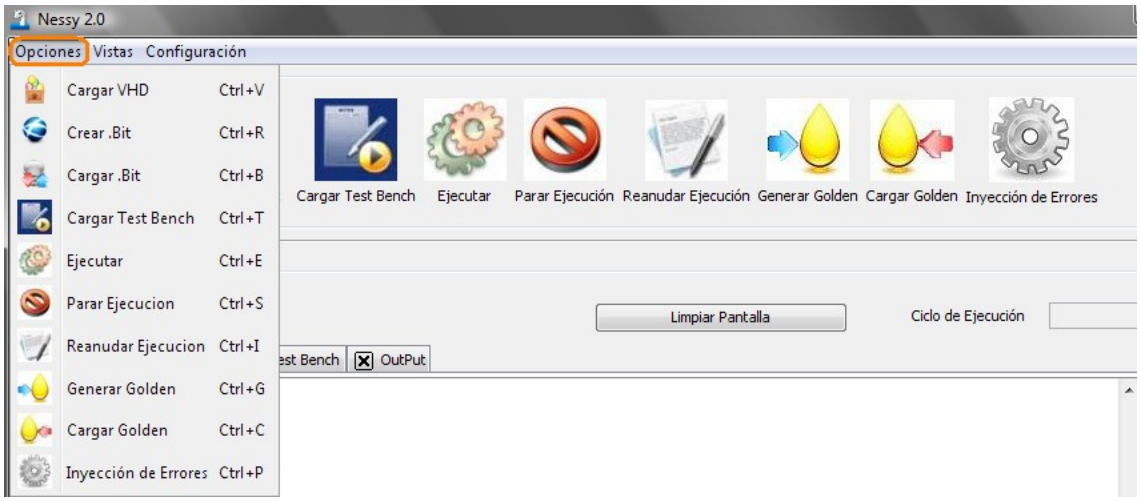

Figura Ap3.3: Desplegable Opciones

Desplegable Vistas

Desde el desplegable vistas, ilustradas en la Figura Ap3.4 se puede seleccionar de entre las vistas disponibles cuál es la que deseamos visualizar. Si la vista se encontrara cerrada se abriría y se seleccionaría. Si por el contrario se encuentra abierta, simplemente se activa como la vista seleccionada, con su pestaña correspondiente.

Tenemos 4 vistas disponibles en la ventana principal:

1. **Entity VHDL:** En ella mostramos la entidad VHDL del archivo top con el que estamos trabajando.
2. **Cargar** : Mostramos la salida generada al cargar un archivo .bit en nuestra aplicación
3. **Test Bench :** Mostramos el banco de pruebas, de forma editable, para que el usuario pueda modificarlo. Se ejecutará este banco de pruebas si previamente seleccionamos la opción de Cargar Test Bench en Pantalla.
4. **Output :** Se muestra la salida generada por la última ejecución.

Ejemplo:

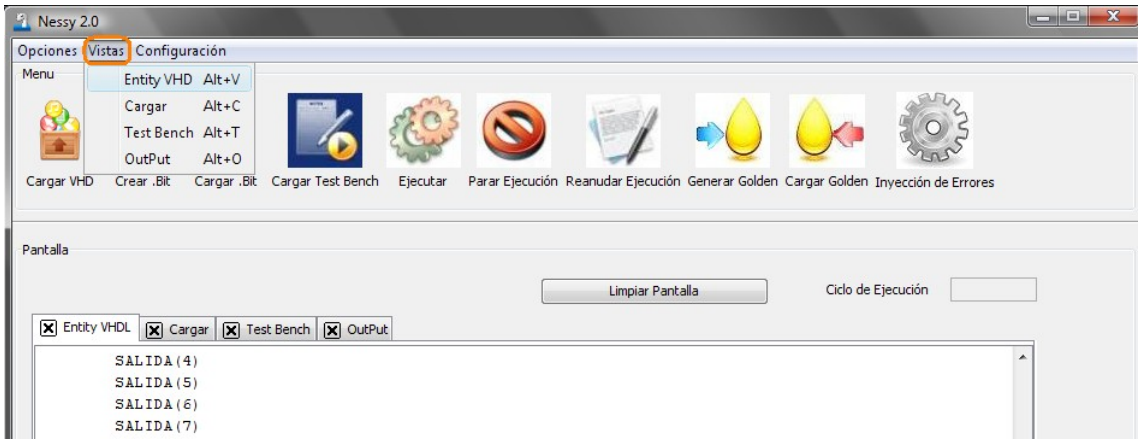

Figura Ap3.4: Desplegable Vistas.

Al pulsar en Output aparece la nueva vista, como se puede apreciar en la Figura Ap3.5. En este caso no había ninguna vista abierta.

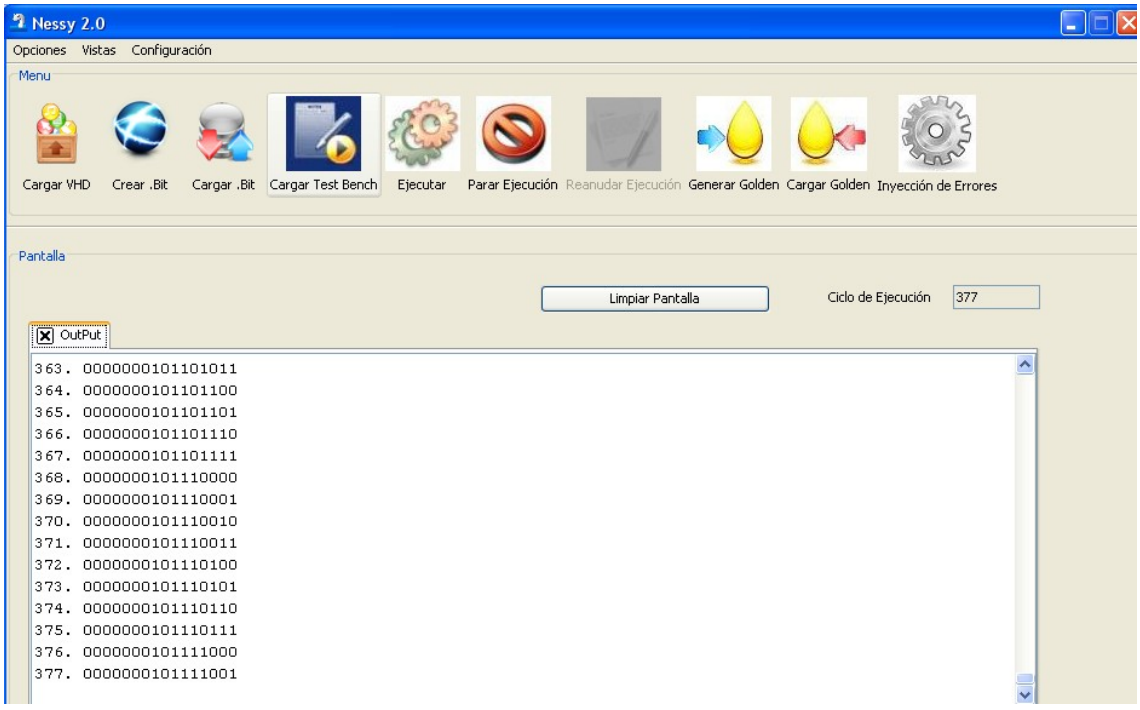

Figura Ap3.5: Vista Output.

El desplegable Configuración

En este menú podremos establecer la carpeta donde se encuentra instalada la aplicación Xilinx. Es

importante que al seleccionarla sea la carpeta raíz de la herramienta Xilinx porque si no, no funcionará correctamente la aplicación. Tenemos dos opciones de configuración, que son configurar Nessy y cargar fichero de ejecución, como se ve en la Figura Ap3.6.

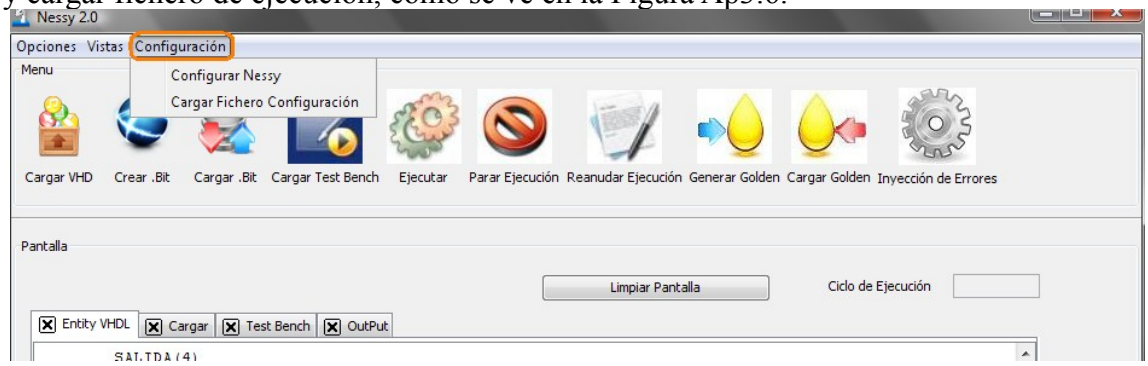

Figura Ap3.6: Desplegable configuración

Configurar Nessy :

Se abre una ventana emergente (ver Figura Ap3.7), en cual podremos seleccionar el fichero raíz donde encuentra la aplicación Xilinx instalada. Deberemos seleccionarla dando al botón de selección y pulsando sobre la carpeta elegida.

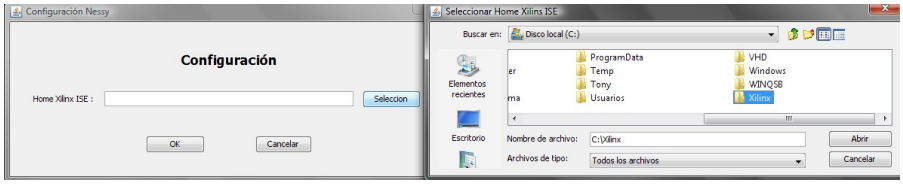

Figura Ap3.7: Configuración de Nessy

Si tuviéramos ya una dirección de Xilinx establecida, ésta aparecería escrita al abrir la ventana. Una vez elegida la nueva dirección bastaría con pulsar OK para guardarla.

Cargar Fichero de Ejecución:

Se abre una ventana para seleccionar el fichero .properties para establecer las propiedades de nuestra aplicación (ver Figura Ap3.8). Si se seleccionara un fichero que no tuviera el campo HomeXilinx daría el aviso que se muestra en la Figura Ap3.9.

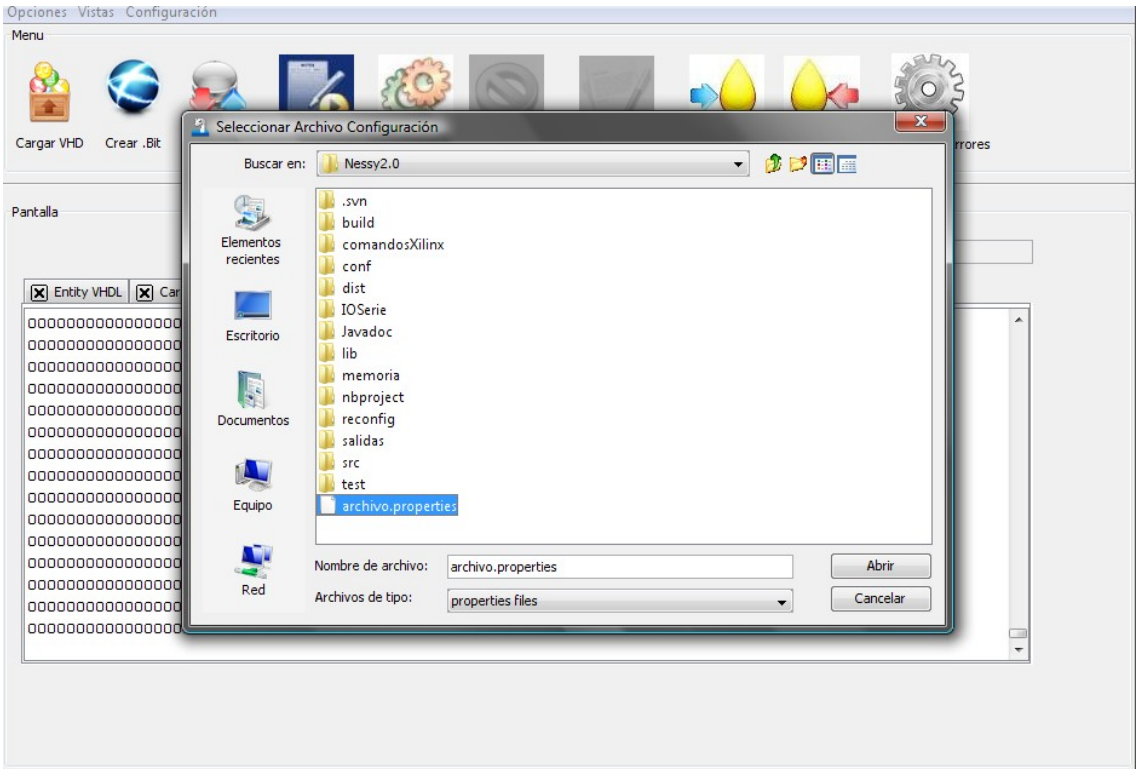

Figura Ap3.8: Cargar fichero de configuración

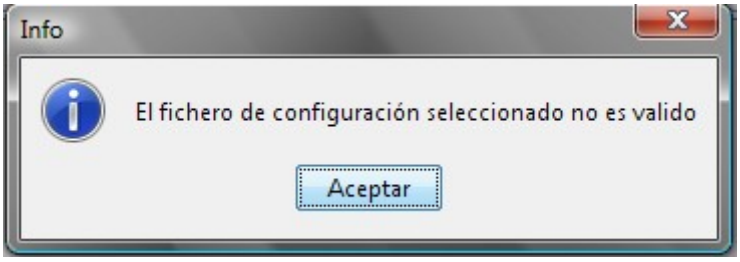

Figura Ap3.9: Error al cargar configuración desde archivo .properties

Barra de botones

Desde ella, como se ve en la Figura Ap3.10, se controlan los procesos más importantes de la herramienta. Al comenzar la aplicación solo estará activo el primer botón (cargar VHD), porque para trabajar correctamente es necesario tener bien definida la entidad con la que deseamos comunicarnos.

Figura Ap3.10: Botonera.

Cargar VHDL 

Desde aquí cargamos los ficheros de tipo VHDL del proyecto con el que queramos trabajar. Al seleccionar esta opción se abrirá una ventana emergente, como se muestra en la Figura Ap3.11. Si nuestro circuito solo contiene un archivo VHDL deberemos seleccionar Cargar un VHDL y este será el archivo Top.

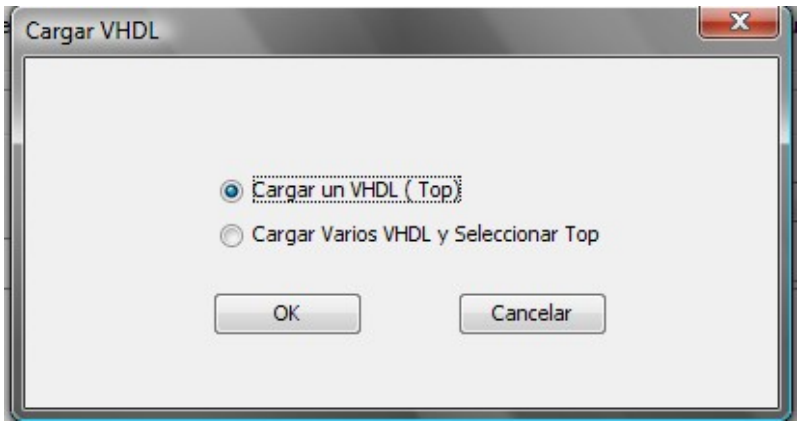

Figura Ap3.11: Opciones de Carga de VHDL

Si por el contrario, queremos seleccionar varios archivos utilizaremos la segunda opción. En ella podremos añadir todos los ficheros VHDL que deseemos, tanto de la misma carpeta como de carpetas distintas. Podemos añadir cuantas veces deseemos. Es importante tener seleccionado un archivo como Top, y solamente uno. En caso de no elegir ninguno, no podremos continuar ya que el proyecto solo puede tener un fichero Top, tal y como como se ve en la Figura Ap3.12.

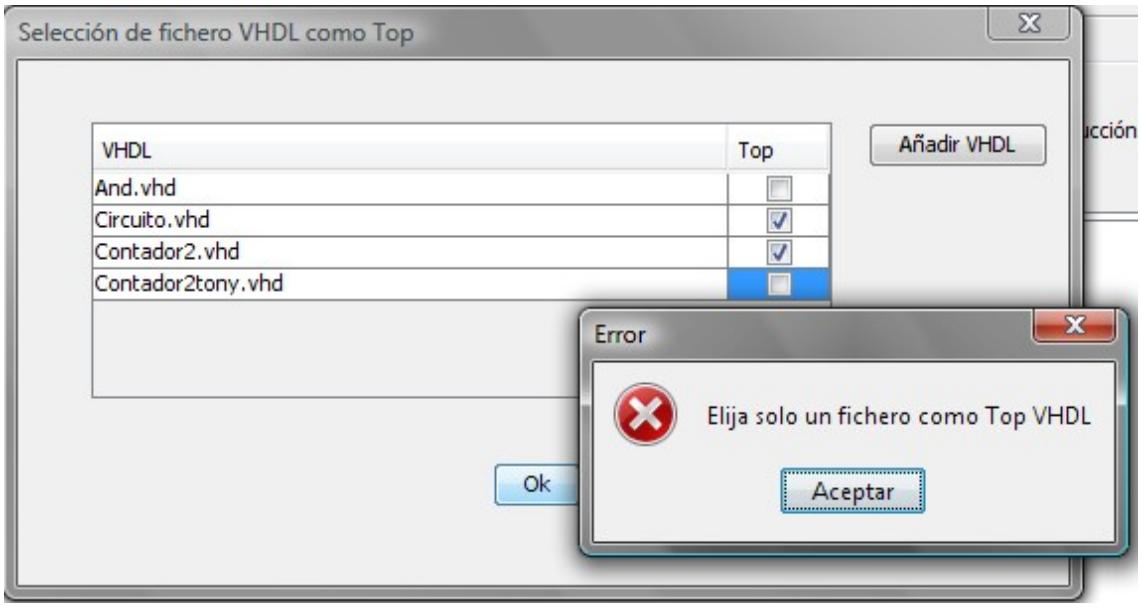

Figura Ap3.12: Error en la selección del arhivo TOP.

Una vez seleccionados los ficheros necesarios se cargará el archivo Top en la Vista Entity VHDL, para la cómoda visualización de la entidad principal.
El último archivo Top cargado correctamente se mostrará en la parte inferior de la ventana (ver Figura Ap3.13).

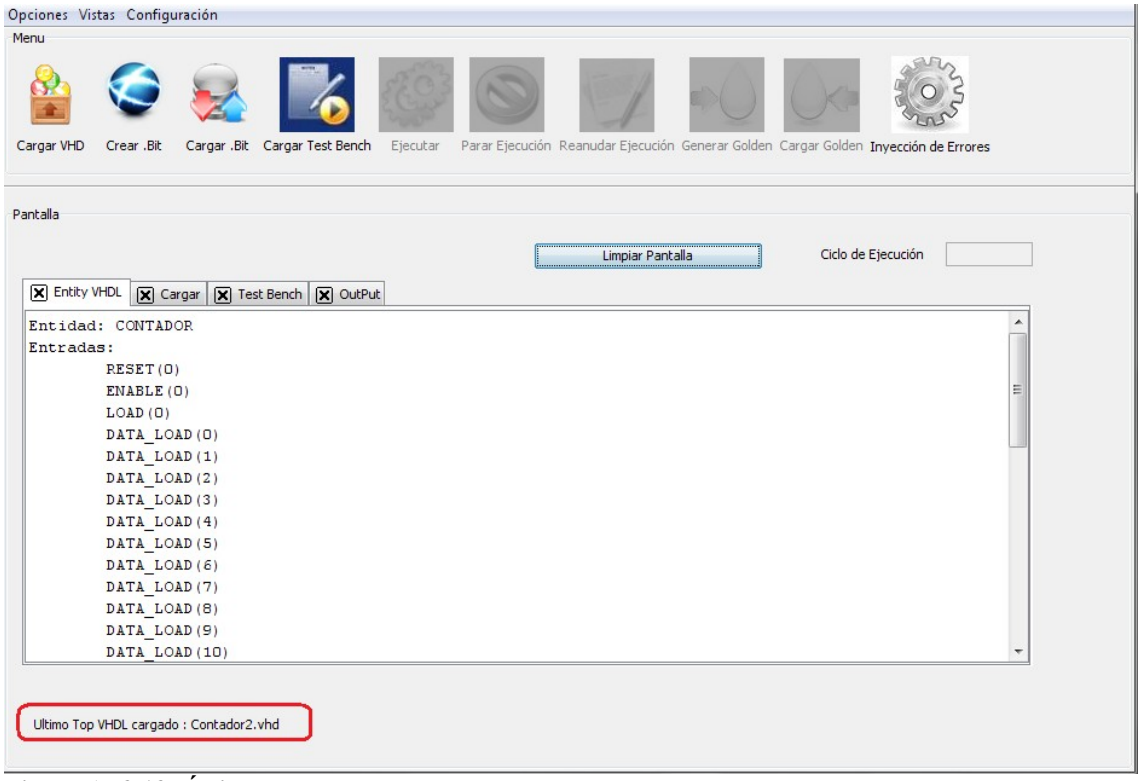

Figura Ap3.13: Último VHDL cargado.

Crear .Bit 

Este botón aparece deshabilitado hasta que no hemos pulsado Cargar VHDL y hemos definido la entidad correctamente.

El primer paso al pulsar este botón es escoger dónde deseamos guardar el fichero .Bit y el nombre que tendrá el fichero, como se indica en la Figura Ap3.14. El fichero .Bit se generará a basándose en la entidad que hemos cargado y siempre que no aparezca ningún error a la hora de generarlo.

Este proceso es lento porque realiza una acción similar a la de crear el .bit desde Xilinx. Se abrirá una consola de comandos (Figura Ap3.15) que no se cerrará por si se desea consultar alguna etapa de la creación.

Internamente generamos un .bit creando un proyecto como haríamos en Xilinx, añandiéndole además los archivos Rx.vhd, Tx.vhd y un archivo que será TOP de todos. Con estos nuevos archivos proporcionamos una comunicación correcta con la FPGA siguiendo el protocolo del puerto serie. No importa el circuito que se declare como Top en nuestra herramienta siempre que no sobrepase las 32 entradas y 32 salidas.

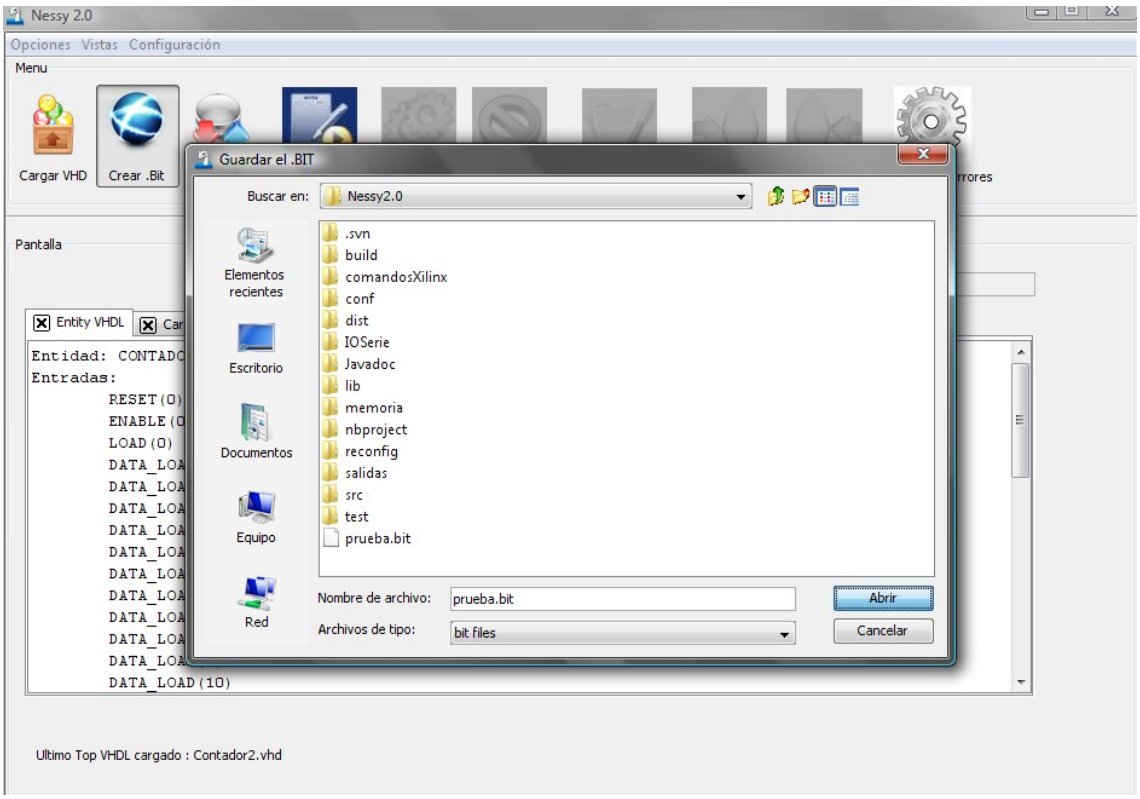

Figura Ap3.14: Guardar el archivo .bit que vamos a crear.

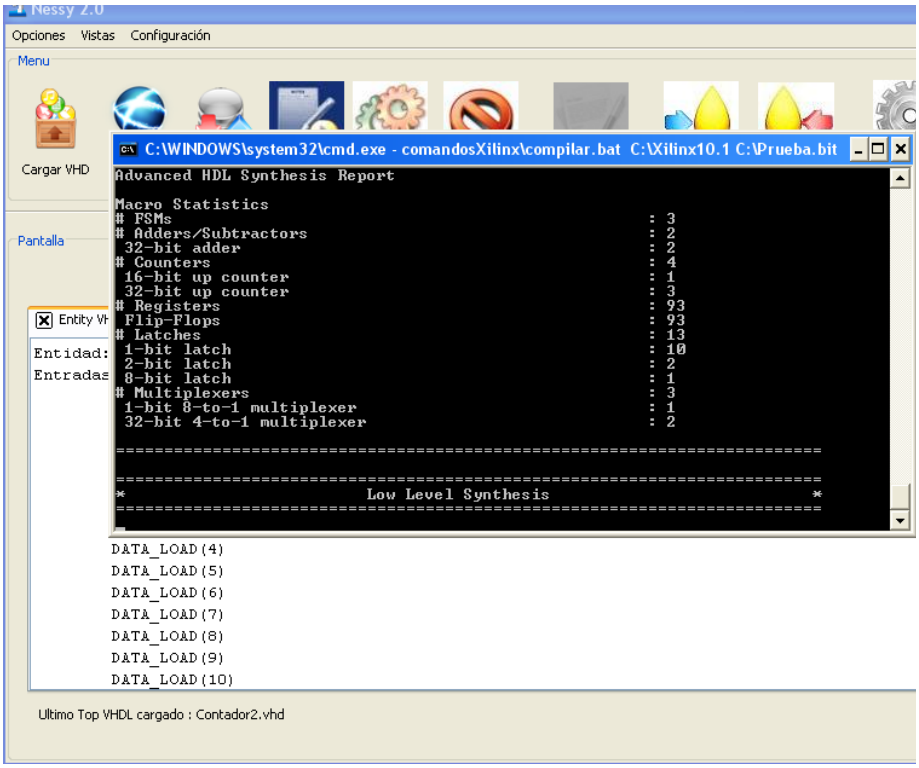

Figura Ap3.15: Creación del archivo .bit

Cargar .Bit 

Desde esta opción podemos cargar el archivo .bit que elijamos. Para ello se abrirá una ventana en la que elegiremos el fichero (Figura Ap3.16). Esta opción está disponible desde que tenemos cargados correctamente los ficheros VHDL, porque podría ocurrir que ya estuviera generado el .BIT que queremos generar, y por tanto no tendríamos necesidad de generarlo de nuevo.

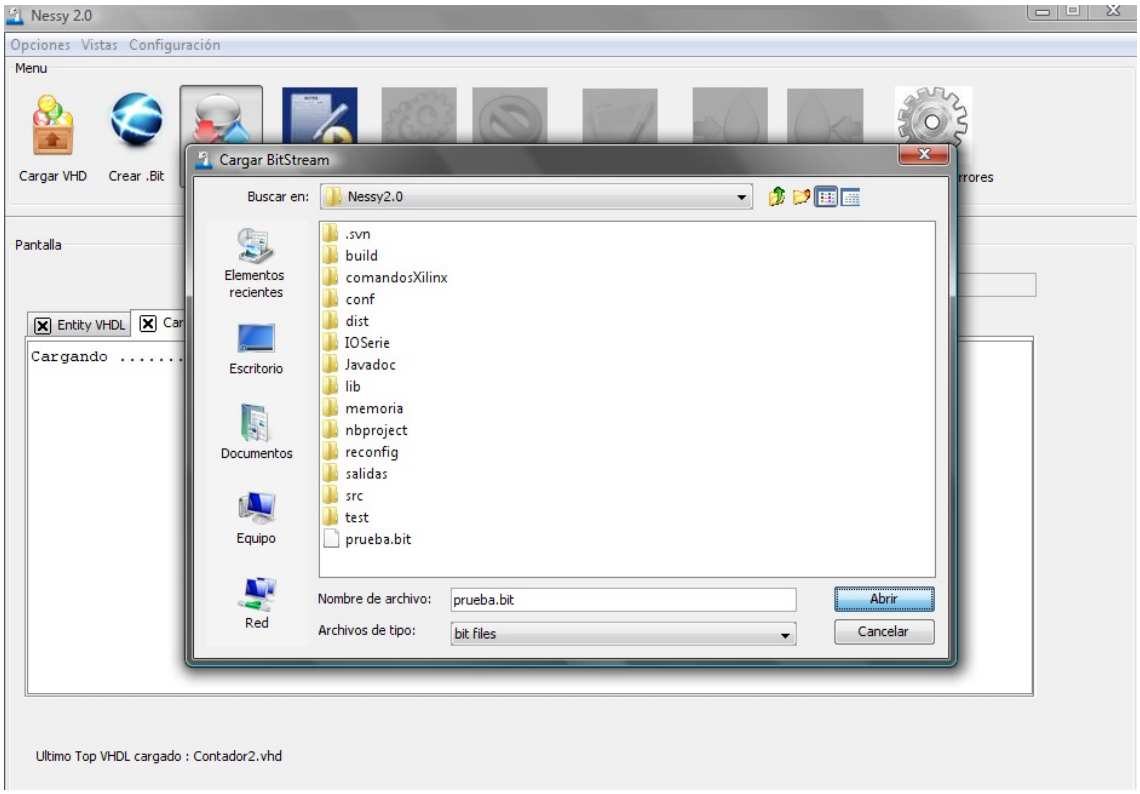

Figura Ap3.16: Carga de un archivo .bit

La salida que genere este proceso se mostrará por la vista Cargar y podremos ver si hemos tenido éxito en la carga del fichero en la FPGA, como en el caso de la Figura Ap3.17.

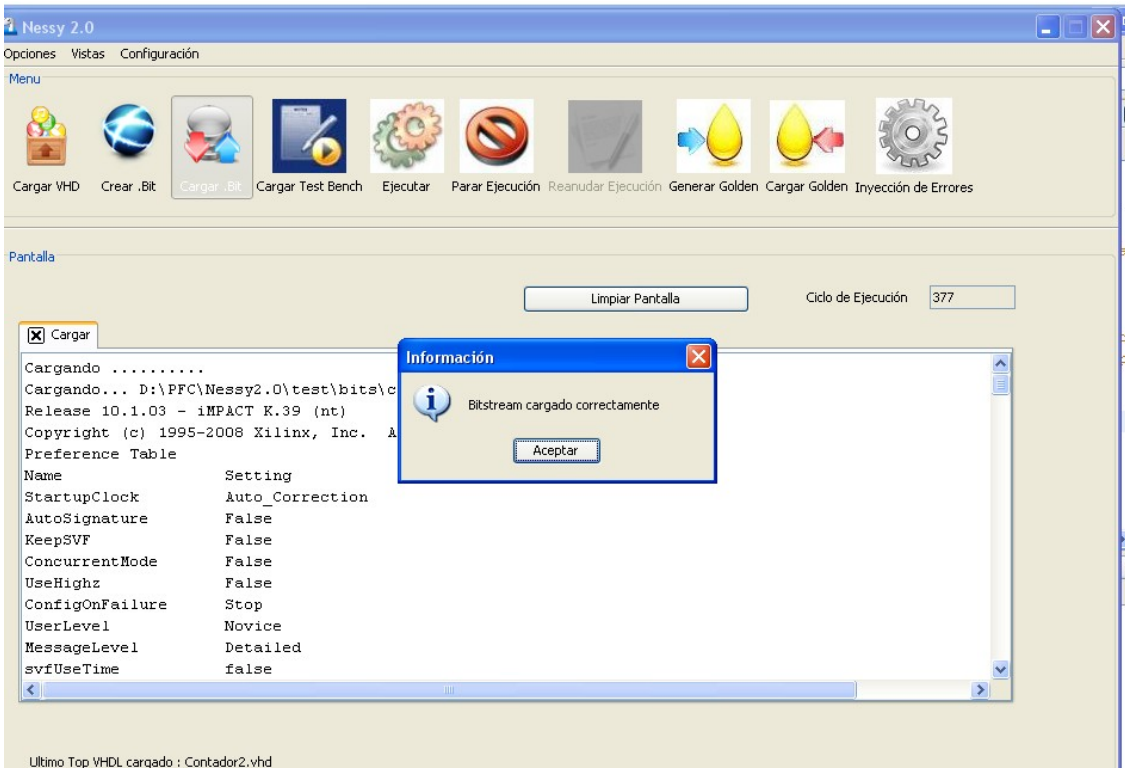

Figura Ap3.17: Carga correcta de un archivo .bit

En la Figura Ap3.18 podemos observar, que en la parte inferior de la ventana principal se encuentra el último archivo .bit que hemos cargado en la aplicación.

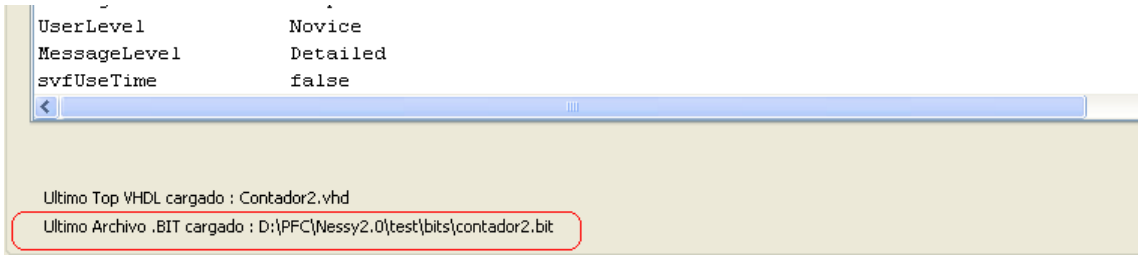

Figura Ap3.18: Detalle del último archivo .bit cargado.

Importante: el fichero .BIT que se cargue tiene que cumplir dos restricciones **para la correcta ejecución posterior**. La primera es que dicho fichero haya sido generado mediante el propio interfaz de usuario (con el botón Crear .bit). La segunda es que el fichero .BIT se corresponda a la entidad que se tiene cargada en ese momento, ya que si no, obtendremos resultados inesperados a la hora de ejecutar. Para ello, al generarlos, recomendamos nombrar a los ficheros .BIT con un nombre parecido al de la entidad a la cual se corresponden. Otro tipo de ficheros .Bit podrán ser cargados pero producirán resultados de ejecución inesperados.

Cargar Test Bench 

En este caso, también esta opción de trabajo, esta habilitada desde que tenemos cargados correctamente los ficheros VHDL, porque puede que ya tengamos cargado en la FPGA el fichero .bit con el que deseamos trabajar.
Al pulsar el botón, se nos ofrecen dos opciones de trabajo, la carga del Test Bench en Pantalla, o desde Fichero y que comience la ejecución (ver Figura Ap3.19):

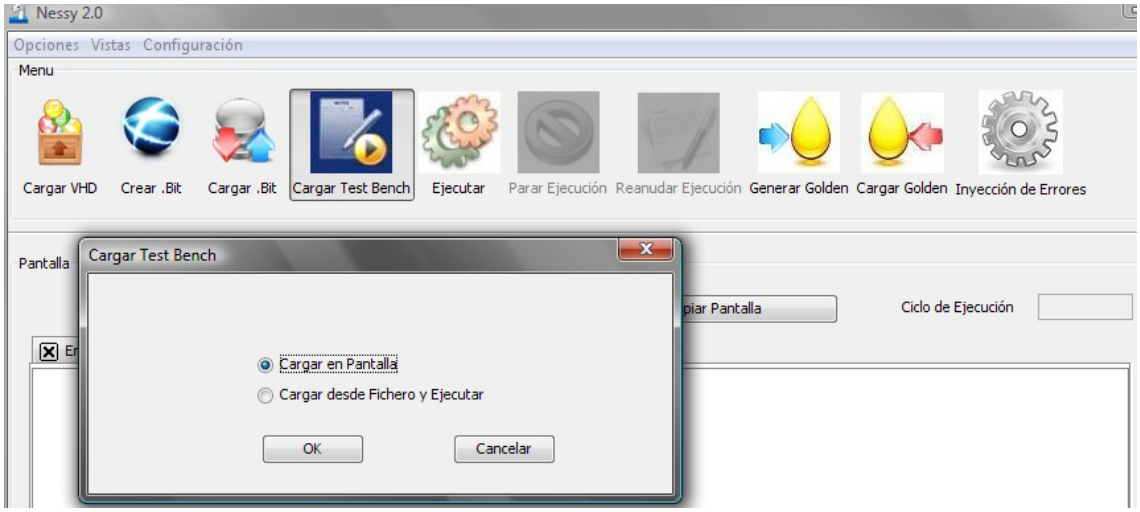

Figura Ap3.19: Opciones de la carga de un Test Bench.

Cargar en Pantalla: Al pulsar la opción Cargar en Pantalla, tendremos que seleccionar el archivo .txt con el repertorio de datos que deseamos enviar a la FPGA, y se nos cargará temporalmente en la vista Test Bench. Esta vista es editable por lo que se puede modificar los datos cuantas veces deseemos. Si tenemos éxito en la carga aparecerá un mensaje como el de la Figura Ap3.20.

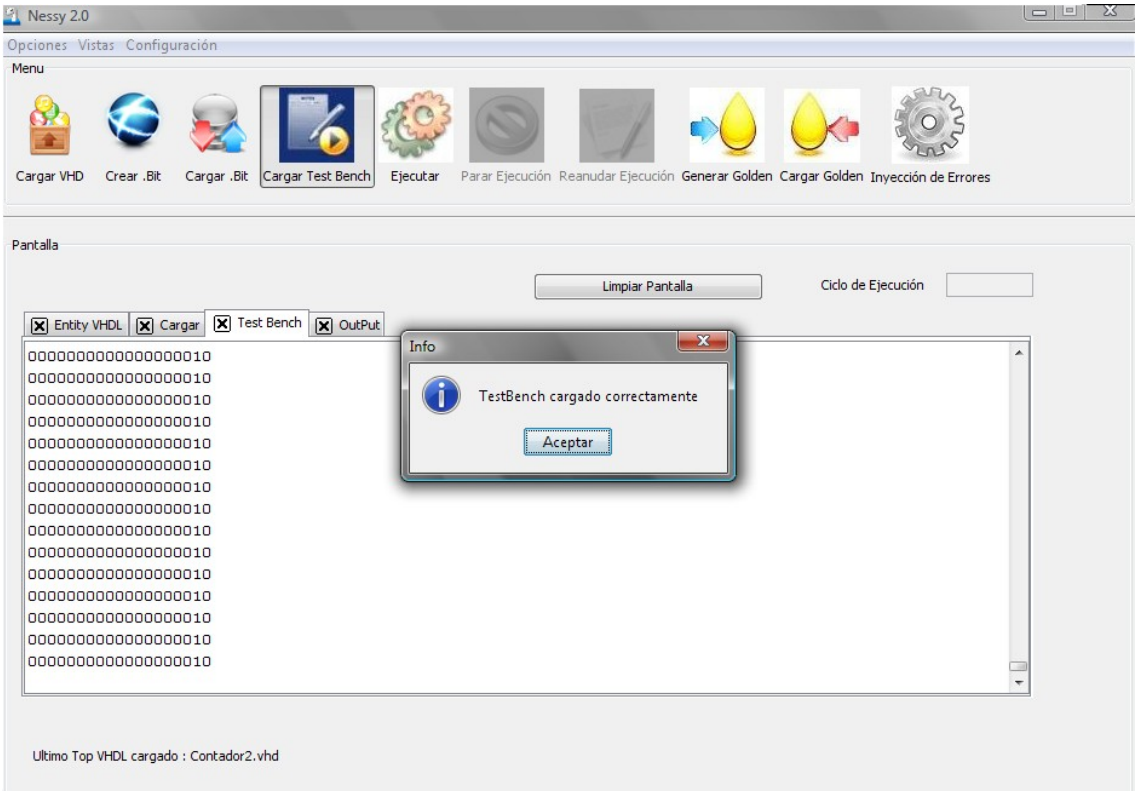

Figura Ap3.20: Cargar Test Bench en Pantalla con éxito.

Si el número de datos a enviar supera las 250.000 líneas, no se podrá ejecutar esta opción, por lo que deberemos seleccionar Cargar desde Fichero y Ejecutar, y nos aparecerá el error de la Figura Ap3.21.

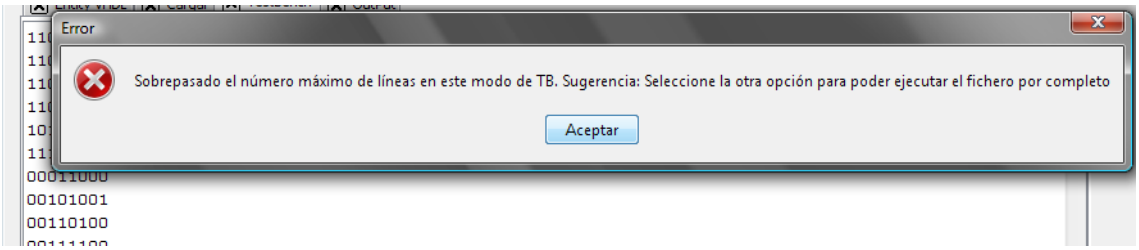

Figura Ap3.21: Error al cargar en Pantalla por sobrepasar el límite de instrucciones.

Cargar desde Fichero y Ejecutar: Con esta opción al seleccionar el archivo de banco de pruebas elegido, comenzará la ejecución siempre que el formato de los datos que contenga ese fichero esté en concordancia con la entidad Top que hayamos elegido, en caso contrario mostrará el error de la Figura Ap3.22.

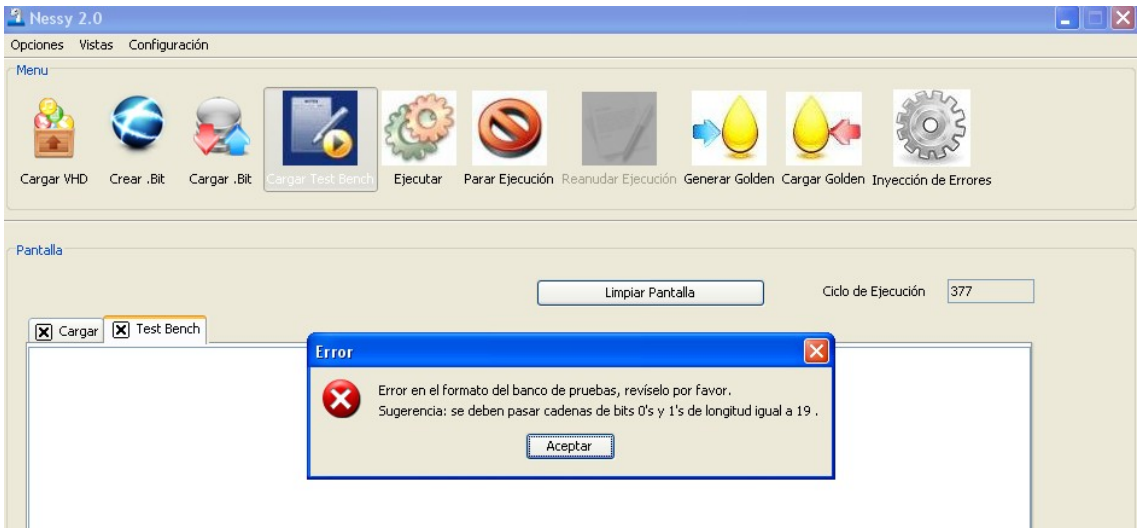

Figura Ap3.22: Error porque el formato del fichero no se corresponde con la entidad.

Para la generación correcta de un TestBench, ya sea cargado en pantalla o desde fichero, se ha de respetar el formato consistente en cadenas binarias por linea, cada cual se corresponde con un dato y tiene que tener la misma longitud que el número de entradas que tengamos definidas en el archivo VHDL que hemos cargado como TOP.
La última entrada declarada en la entidad TOP se corresponde con el bit más significativo, el siguiente bit se correspondería con la penúltima entrada declarada, y así sucesivamente hasta que llegamos a la primera entrada declarada que sería el bit menos significativo.
Ejemplo :

```
entity MiEntidadInventada is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           z : out STD_LOGIC);
end MiEntidadInventada;
```

Aquí solo tenemos tres entradas, deberíamos generar cadenas de longitud 3.

Para a = 0, b = 1, c = 1 deberíamos enviar 110

Para a = 1, b = 1, c = 0 deberíamos enviar 011

Ejecutar 

Esta opción esta habilitada, en el momento que hemos definido el banco del pruebas con el que queremos trabajar. Al pulsarlo se comienzan a enviar los datos que tengamos en la vista del Test Bench si hemos elegido cargar los datos en pantalla, o los cogerá del fichero que le hallamos definido, y se empezarán a visualizar en la vista Output como se aprecia en la Figura Ap3.24.
En el caso de elegir cargar los datos en pantalla si el formato no es el correcto nos aparece el mensaje de la Figura Ap3.23 y deberemos modificar el Test Bench que tenemos en la solapa y volver a ejecutar.
Es importante que el formato de los datos sean cadenas de números binarios, con una longitud igual a la suma de las entradas del archivo Top que hemos definido al cargar la Entidad.
La primera entrada que tenemos definida se corresponde con el bit menos significativo de la cadena y la última entrada con el bit más significativo de la entrada. Esto es importante tenerlo en cuenta cuando definamos los datos que queremos mandar.
Si alguna de las cadenas contiene un carácter que no se un número binario o la cadena tenga una longitud distintas del número de entradas, saltará un aviso y no se ejecutará nada (Figura Ap3.23).
La salida que se genera se compara con la última salida Golden que se guarda en un fichero de texto. Tanto si la salida de la ejecución actual coincide con la Golden como si no se mostrará un mensaje de información. La salida que se genera al pulsar este botón se guarda en un archivo de texto llamado Salida.txt dentro de la carpeta de salidas.

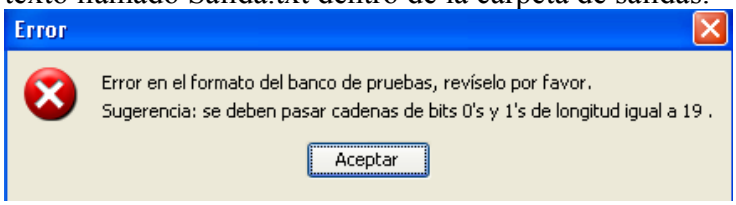

Figura Ap3.23: Error al ejecutar con Test Bench cargado en Pantalla.

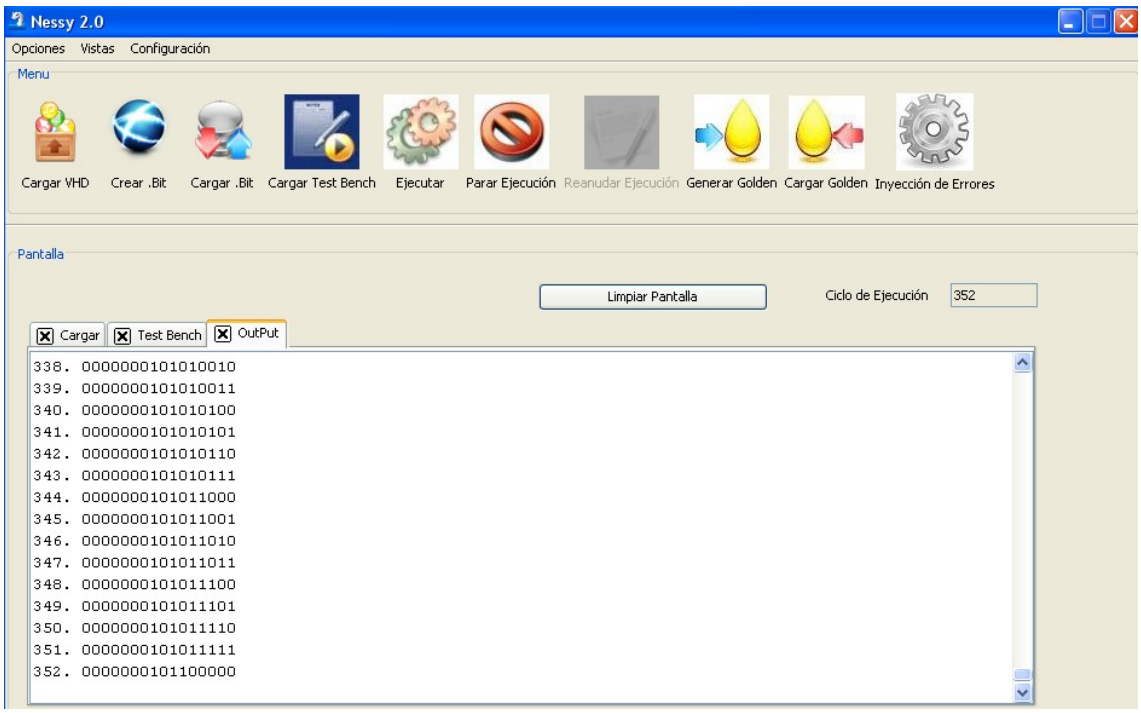

Figura Ap3.24: Ejecución correcta

Durante la ejecución se puede observar el número de datos enviados por el que va el proceso. Al pulsar ejecutar, y mientras dure la ejecución se habilita el botón Parar Ejecución para detener el proceso.

Parar Ejecución 

Este botón se habilita mientras dura la ejecución de un banco de pruebas. Se detiene el envío de datos y se deja de recibir. Podemos observar en el dato que nos hemos quedado.
El parar la ejecución no afecta en absoluto a la salida que se produzca, al menos que se modifique el archivo .bit cargado en la placa o el fichero con los datos que se están enviando a la FPGA.
Si la salida actual que se está generando no coincide con la salida Golden al pulsar este botón nos alertaría ya de este hecho (como ocurre en la Figura Ap3.25), indicándonos también el dato que causó la diferencia.

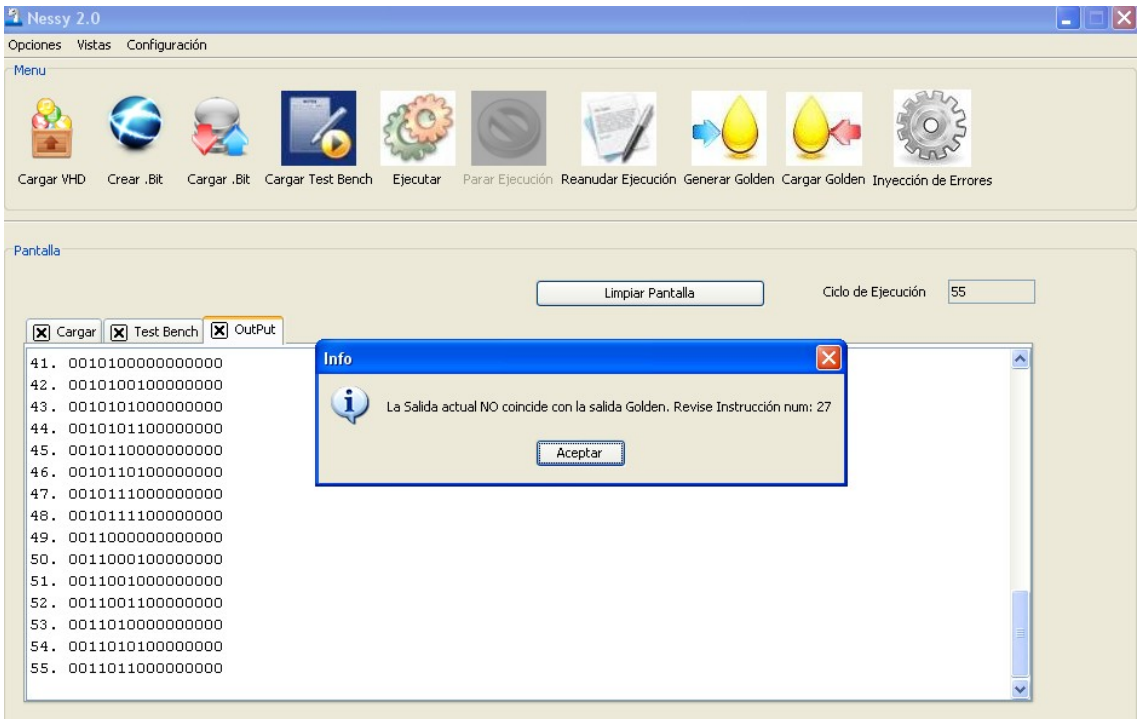

Figura Ap3.25: Parar la ejecución, con distinta salida.

Al pulsar este botón se deshabilita el propio botón y habilita el botón de reanudar ejecución. Vuelve estar habilitado al volver al pulsar ejecución, o al pulsar reanudar ejecución.

Reanudar Ejecución

Este botón esta habilitado cuando hemos parado la ejecución, y permite continuar la ejecución en el punto donde se estaba antes de pulsar parar ejecución.
Este botón solo esta habilitado mientras dura la ejecución y después de haber parado la ejecución. El efecto al pulsarlo es que habilita parar ejecución y prosigue el envío de datos a la FPGA, como ocurre, en la Figura Ap3.26.

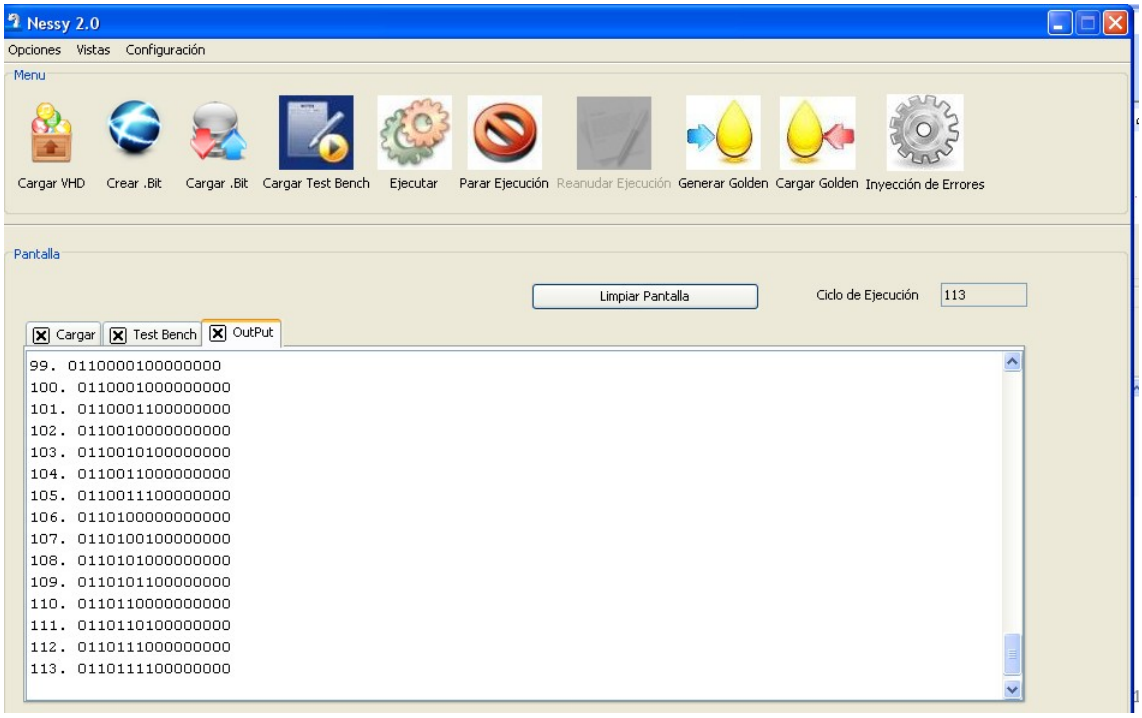

Figura Ap3.26: Reanudar Ejecución.

Generar Golden 

Este botón queda habilitado después de que tengamos definido el conjunto de datos del banco de pruebas que deseamos enviar a la placa. Su funcionalidad es la de una ejecución pero guarda la salida en Golden.txt y será el fichero de referencia para comparar el resto de ejecuciones y ver si ha habido algún error entre las distintas ejecuciones.

Cargar Golden 

Se encuentra habilitado al definir el cargar el banco de pruebas. Al pulsar este botón se abrirá una ventana para elegir el fichero que a partir de ese momento será nuestra salida Golden con la que comparemos todas las ejecuciones.

Inyección de Errores 

Este botón se encuentra habilitado al tener definida la entidad con la que vamos a trabajar. La funcionalidad es ir modificando bit a bit el archivo .bit que elijamos al principio para ir viendo como afecta la modificación de ese bit en la salida.

Al pulsar el botón, lo primero será indicar el número de iteraciones que deseamos que haga (Ver Figura Ap3.27).

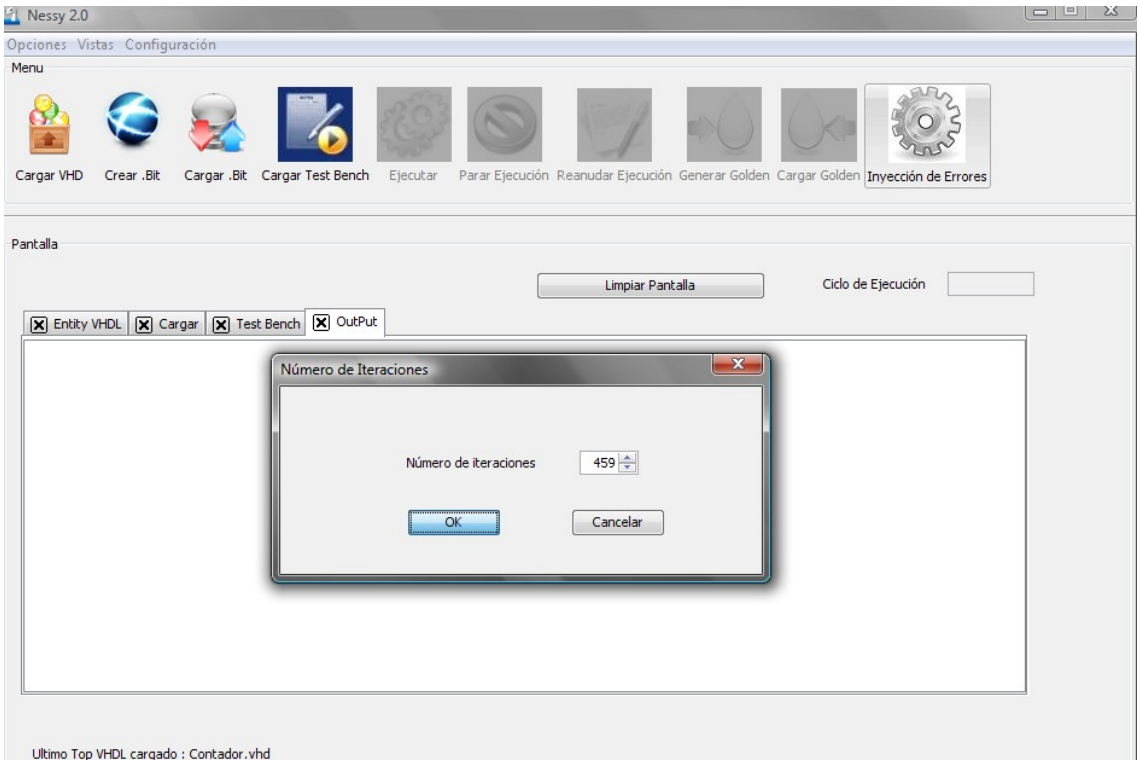

Figura Ap3.27: Selección del número de iteraciones.

Lo siguiente que tenemos que elegir es el archivo .bit (Figura Ap3.28) que vamos a tomar como referencia y sobre el que haremos las modificaciones. Posteriormente tendremos que seleccionar el fichero de banco de pruebas (Figura Ap3.29) con el repertorio conjunto de datos que se van a enviar.

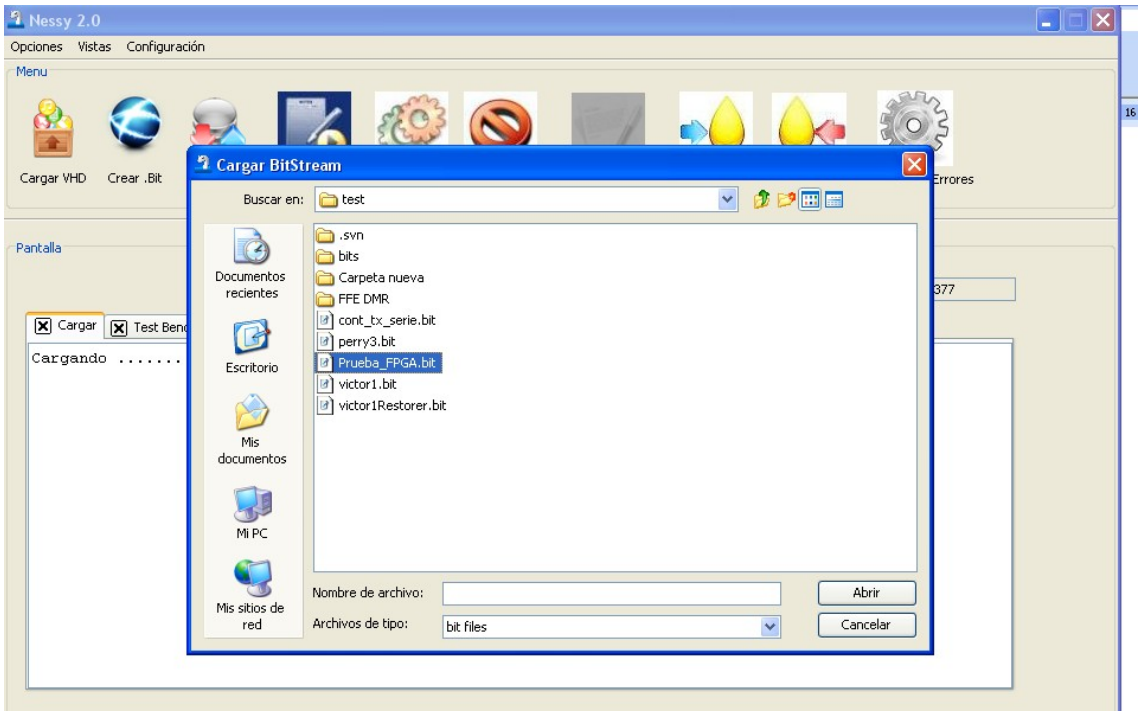

Figura Ap3.28: Inyección de Errores. Cargar .bit de trabajo.

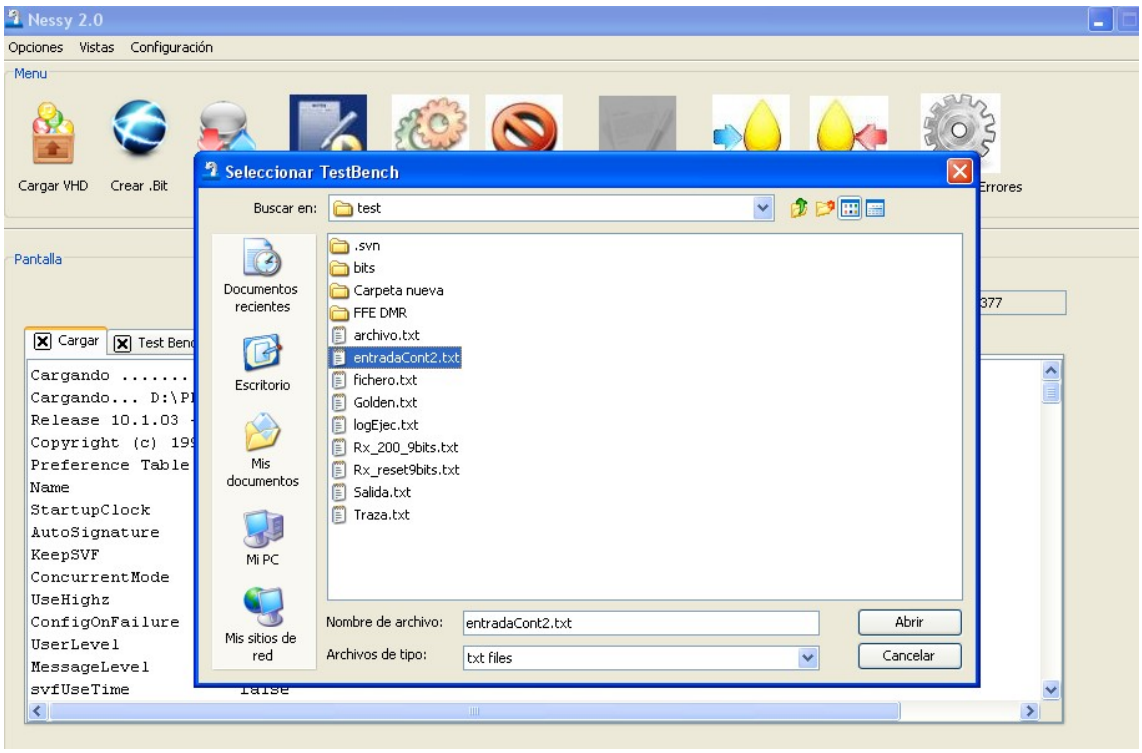

Figura Ap3.29: Inyección de Errores. Cargar Test Bench.

En un primer momento se hará una ejecución sin modificar el archivo para generar la salida Golden, y así, cuando vayamos modificando el archivo cargado en la FPGA ver como ha afectado a la ejecución.

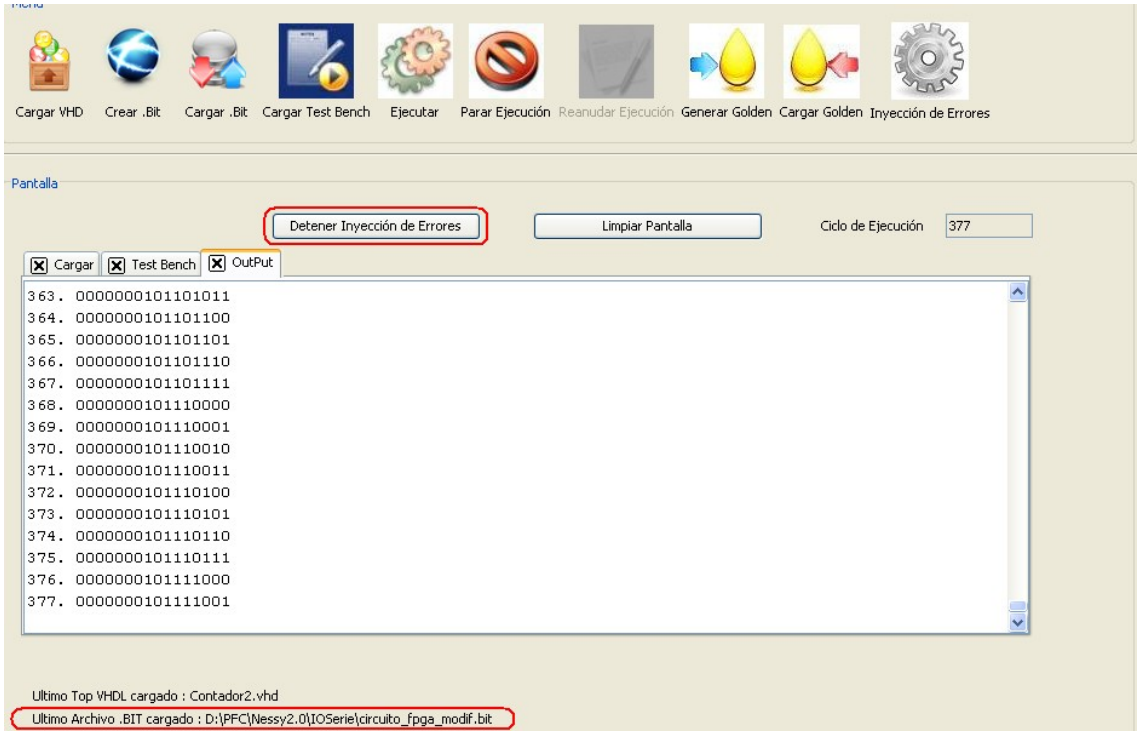

Figura Ap3.30: Inyección de Errores.

La salida, que genera este proceso se guarda en un fichero logEjec.txt, donde se puede revisar en que punto la ejecución fue distinta porque el bit modificado afecto a la salida generada.
Es importante tener en cuenta que como es proceso largo no se muestran ventanas emergentes con

el resultado de la comparación entre la salida Golden y la última ejecución que se ha producido por comodidad para el usuario (Figura Ap3.30).
Al pulsar este botón se habilita un botón (Detener Inyección de Errores) debajo de la barra de botones que permite abortar la inyección de errores, por si el usuario decide terminar el proceso.

Toda la salida y diferencias generadas por la inyección de errores se guardan en un fichero de texto, para saber como ha afectado la modificación del mapa de bits a la salida. Dicho fichero se guarda en *salidas/logEjec.txt*. Un ejemplo de dicho fichero se muestra en la Figura Ap3.31.

```
0000000101101111
0000000101110000
0000000101110001
0000000101110010
0000000101110011
0000000101110100
0000000101110101
0000000101110110
0000000101110111
0000000101111000
0000000101111001
La Salida actual coincide con la Golden

Cargando BIT: D:\PFC\Nessy2.0\IOSerie\circuito_fpga_modifRestorer.bit


Modificando FRAME: 18601 BIT: 4
Ejecutando: cmd.exe /K java -jar Virtex_II_Partial_Reconfiguration.jar -i D:\PFC\Nessy2.0\test\Prueba_FPGA.bit -o D:\PFC\Nessy2.0\
Cargando BIT: D:\PFC\Nessy2.0\IOSerie\circuito_fpga_modif.bit
Ejecutando...
0000000000000001
0000000000000010
0000000000000011
0000000000000100
0000000000000101
```

Figura Ap3.31 Ejemplo de fichero log de inyección de errores

Más detalles a tener en cuenta:

En la primera ejecución de la aplicación se abrirá la ventana para definir la ruta HomeXilinx y hasta que no la definamos no podremos acceder a la ventana principal.
El botón de Limpiar Pantalla situado debajo de la barra de botones sirve para limpiar el contenido de la vista que actual.
El último .bit y el último archivo Top utilizados durante el uso de la aplicación se detallan en la parte inferior de la ventana principal, como se puede ver en la Figura Ap3.32.

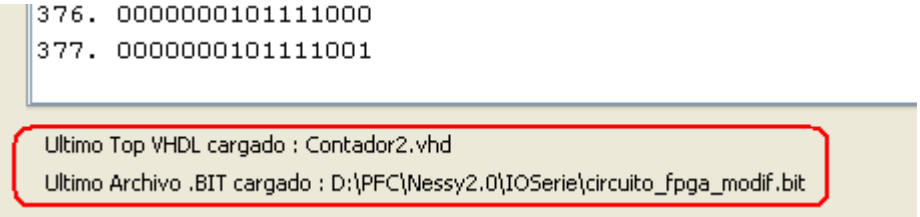

Figura Ap3.32: Detalle del último .bit y VHDL (top) cargardos.

Log de la aplicación

Durante la ejecución de la aplicación, se irá generando un fichero log en el que se indique todas las acciones que el usuario ha ido realizando, así como posibles errores que hayan podido producirse tanto en la ejecución actual como en ejecuciones anteriores. De esta manera si se ha experimentado un resultado anómalo o inesperado, se podrá consultar este fichero para ver qué ha podido suceder. Dicho archivo se crea en la ruta -> ***C:\aplicacion.log***.

Teclas Rápidas:

Para facilitar la interfaz con el usuario, existe la siguiente combinación de teclas rápidas:

Opciones:

- Cargar VHD → Control + V
- Crear .Bit → Control + R
- Cargar .Bit → Control + B
- Cargar Test Bench → Control + T
- Ejecutar → Control + E
- Parar Ejecución → Control + S
- Reanudar Ejecución → Control + I
- Generar Golden → Control + G
- Cargar Golden → Control + C
- Inyección de Errores → Control + P

Vistas:

- Entity VHD → Alt + V
- Cargar → Alt + C
- Test Bench → Alt + T
- Output → Alt + O